МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп’ютерних технологій

Кафедра комп’ютерних наук

**КВАЛІФІКАЦІЙНА РОБОТА**

на тему:

*Програмне забезпечення для автоматичного розпізнавання усної природної мови за допомогою системи штучного інтелекту*

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп’ютерні науки

Освітня програма Комп’ютерні науки

Виконав: студент групи МгІТ-1-23

Тарасенко Владислав Сергійович

Науковий керівник

к.т.н. доц. Гольдбрег М.І.

Рецензент

д.ф-м.н. проф. Краснитський С.М.

Київ 2024

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет Мехатроніки та комп'ютерних технологій

Кафедра Комп’ютерних наук

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп’ютерні науки

Освітня програма Комп’ютерні науки

**ЗАТВЕРДЖУЮ**

Завідувач кафедри

(абревіатура кафедри)

(підпис) (Власне ім’я та ПРІЗВИЩЕ)

« » 20 р.

**ЗАВДАННЯ**

**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**

*Тарасенку Владиславу Сергійовичу*

1. Тема кваліфікаційної роботи *Програмне забезпечення для автоматичного розпізнавання усної природної мови за допомогою системи штучного інтелекту*

Науковий керівник роботи *Гольдберг Мар’яна Ігорівна , к.т.н. доц.*

затверджені наказом КНУТД від «*03*» *вересня 2024 року* № *188-уч*

1. Вихідні дані до кваліфікаційної роботи (проєкту) *Розробка кафедри комп’ютерних наук. Програмне забезпечення для автоматичного розпізнавання усної природної мови за допомогою системи штучного інтелекту.*
2. Зміст кваліфікаційної роботи (проєкту) (перелік питань, які потрібно опрацювати) *РОЗДІЛ 1 математичний огляд; РОЗДІЛ 2 створення моделі; РОЗДІЛ 3 створення демо, порівняння результатів моделі, наступні кроки.*
3. Дата видачі завдання *08.2024р*

5.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапу  кваліфікаційної роботи (проєкту) | Орієнтовний термін виконання | Примітка про виконання |
| 1 | Вступ | 09.10.2024 |  |
| 2 | Розділ 1. Математичне забезпечення | 12.10.2024 |  |
| 3 | Розділ 2. Алгоритмічне забезпечення | 20.10.2024 |  |
| 4 | Розділ 3. Програмне забезпечення | 25.10.2024 |  |
| 5 | Висновки | 25.10.2024 |  |
| 6 | Оформлення (чистовий варіант) | 30.10.2024 |  |
| 7 | Подача кваліфікаційної роботи (проєкту) науковому керівнику для відгуку (за 14 днів дозахисту) | 30.10.2024 |  |
| 8 | Подача кваліфікаційної роботи (проєкту) для рецензування (за 12 днів дозахисту) | 30.10.2024 |  |
| 9 | Перевірка кваліфікаційної роботи (проєкту) на наявність ознак плагіату (за 10 днів до захисту) |  |  |
| 10 | Подання кваліфікаційної роботи (проєкту) завідувачу кафедри  (за 7днів до захисту) |  |  |

З завданням ознайомлений:

Студент Владислав ТАРАСЕНКО

(підпис) (Власне ім’я та ПРІЗВИЩЕ)

Науковий керівник Мар’яна ГОЛЬДБЕРГ

(підпис

**АНОТАЦІЯ**

**Тарасенко Владислав. Програмне забезпечення для автоматичного розпізнавання усної природної мови за допомогою системи штучного інтелекту.**

Дипломна магістерська робота за спеціальністю 122 - «Комп’ютерні науки». – Київський національний університет технологій та дизайну, Київ, 2024 рік.

Мета роботи – Створення програмного забезпечення для автоматичного розпізнавання усної української мови та генерування транскрипції почутого за допомогою системи штучного інтелекту.

Для реалізації мети поставлені наступні задачі:

* Математичний огляд класичних методів вирішення задачі та методів з використанням ШІ.
* Створення моделі: підбір архітектури, збір даних, тренування, оцінка.
* Створення візуального демо, порівняння результатів моделі, надання наступних кроків для покращення точності моделі.

Загальний обсяг роботи: 124 сторінки, 59 рисунків, 26 посилань, 1 додаток.

*Ключові слова: Pytorch, Transformer, Deep learning Технології: Pytoch, Transformer, Deep learning*

**ANNOTATION**

**Tarasenko Vladyslav. Software for automatic speech recognition using artificial intelligence.**

Master’s degree diplomа paper for the 122 major - «Computer Science». – Kyiv National University of technologies and design, Kyiv, 2024.

Purpose – Creating a software package to do the following task: automatic speech recognition with transcription generation using artificial intelligence approach.

In order to achieve before-mentioned purpose the following goals were set:

* Mathematical overview of classical methods for solving automatic speech recognition And methods involving AI.
* Model creation: choosing architecture, dataset creation, training and validation.
* Creation of visual demo, comparing accuracy results, suggesting next steps to improve accuracy.

Paper’s total size: 124 pages, 59 pictures, 26 links, 1 extra material.

*Keywords: Pytorch, Transformer, Deep learning Technologies: Pytoch, Transformer, Deep learning*

ЗМІСТ

Вступ …7

Аналіз предметної області 9

РОЗДІЛ 1. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ 11

* 1. Обробка природної мови, вилучення ознак. 11
  2. Моделювання… 16
  3. Традиційні методи машинного навчання 17
  4. Методи глибинного навчання… 21

Висновки до розділу 1 43

РОЗДІЛ 2. СТВОРЕННЯ МОДЕЛІ 44

* 1. Створення набору даних для тренування… 44
  2. Архітектура моделі 49
  3. Процес тренування моделі 54
  4. Проблеми локального тренування.

Перенесення у хмарне середовище 67

Висновки до розділу 2 80

РОЗДІЛ 3. ПІСЛЯ ТРЕНУВАННЯ 81

* 1. Аналіз результатів 81
  2. Порівняння з іншими моделями… 93
  3. Демо застосунок… 96
  4. Рекомендації щодо подальшого тренування 108

Висновки до розділу 3 113

ВИСНОВКИ 115

СПИСОК ДЖЕРЕЛ 121

ВСТУП.

Актуальність задачі обробки природної мови (ОПМ) полягає в її здатності розв'язувати широкий спектр практичних проблем та покращувати взаємодію людини з технологіями. ОПМ є невід’ємною частиною повсякденного життя і стає все більш актуальною, оскільки мовні технології застосовуються в різноманітних галузях, таких як роздрібна торгівля (наприклад, у чат-ботах обслуговування клієнтів) і медицина (інтерпретація або узагальнення електронних медичних записів). Розмовні агенти, такі як Alexa від Amazon і Siri від Apple, використовують ОПМ, щоб слухати запити користувачів і знаходити відповіді. Найдосконаліші агенти, такі як GPT-3, який нещодавно було відкрито для комерційних застосувань, можуть генерувати складну прозу на різноманітні теми, а також потужні чат-боти, які здатні вести послідовні розмови. Google використовує ОПМ для покращення результатів пошукових систем, а соціальні мережі, як-от Facebook, використовують його для виявлення та фільтрації ворожнечі.

Спектр використання автоматичного розпізнавання усної мови дуже широкий: створення нотаток з бізнес зустрічей у корпоративному світу, автоматична генерація субтитрів для контенту в індустрії розваг, інтеграція у пристрої для допомоги людям з обмеженими можливостями тощо. Але розпізнавання усної мови потребує величезних знань системи про навколишнє середовище та можливості взаємодіяти з ним. Саме означення змісту слова «розуміти» — одна з головних задач штучного інтелекту. У процесі дослідження обробки природної мови було досягнуто значних результатів, серед яких розробка потужних лексикографічних систем, програм для машинного перекладу, електронних словників та ін. Однак, існує проблема, яка досі не знайшла свого вирішення, вона полягає у самій природі людської мови, а саме у її неоднозначності. Одним із викликів, який виникає у процесі обробки природної мови, можна вважати проблему синонімії, в результаті якої одне поняття може бути вираженим декількома різними словами.

**Мета дослідження**. Вивчення роботи наявних підходів із обробки усної природної мови.

**Завдання дослідження.** Створити модель штучного інтелекту для розпізнавання усної української мови та генерації транскрипції почутого. Порівняти точність новітніх технік штучного інтелекту (ШІ) з більш класичними методами. Оцінити доцільність використання ШІ.

**Об’єкт дослідження.** Технологічні процеси створення моделі глибинного навчання для задачі обробки природної мови.

**Предмет дослідження**. Обробка природних мов, а саме українського мовлення.

**Практична цінність.** Було створено модель, що готова для використання у різних бізнес задачах з автоматичної генерації субтитрів на основі українського мовлення. Модель може послужити основою для створення більш точних моделей під специфічний сценарій використання.

**Елементи наукової новизни** – Мають узагальнюючий характер і містять власні висновки та рекомендації з удосконалення/створення моделей для усного розпізнавання природних мов.

Для досягнення поставленої мети в роботі були поставлені та вирішені наступні завдання: Розглянути існуючі алгоритми для розпізнавання усної мови та генерування транскрипції. Розглянути новітні підходи до задачі з використанням штучного інтелекту. Створити набір даних для тренування моделі. Реалізувати обраний підхід в модель, протренувати модель на достатній кількості даних та протестувати модель за допомогою опорних даних. Порівняти з існуючими моделями з іншої архітектурою. Реалізувати простий веб додаток для демонстрації роботи моделі. Надати рекомендації щодо подальшого покращення створеної моделі.

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.

Автоматичне розпізнавання усної мови значно розвинулося протягом десятиліть завдяки прогресу в алгоритмах, обчислювальній потужності та доступності даних. Ця галузь дослідження бере початок у 1950-х роках, коли працівниками Bell Labs була створена система Одрі (Audrey), яка могла розпізнавати цифри вимовлені одним мовцем. Одрі базувалася на акустико-фонетичному моделюванні. Потім, у 1960-х роках дослідження продовжили в компанії IBM, з їх системою “Взуттєва коробка” (Shoebox), яка могла розпізнати 16 слів. Такі ранні системи як Одрі та Взуттєва коробка значною мірою покладалися на правила, створені вручну, мали обмежений словниковий запас і залежали від голосу мовця.

У 70-х роках був запроваджений алгоритм динамичної трансформації часової шкали, який полягав у порівнянні двох послідовностей (зразків мовлення), методом їх розтягування або стискання. Це давало змогу вибрати найбільш відповідність між послідовностями, навіть якщо вони відрізнялися за швидкістю. Також у цьому століття були створені перші приховані марківські моделі, що заклали основу для статистичних підходів до розпізнавання мовлення.

У 80-х роках приховані марківські моделі були технологією, що домінувала у створенні систем розпізнавання усної мови. Ці моделі використовували ймовірнісні представлення звуків мови та могли обробляти варіації мови краще, ніж попередні системи, засновані на правилах. Компанія IBM розробила систему під назвою Tangora, яка могла розпізнавати до 20 000 слів: значний прогрес у розмірі словникового запасу.

У 1990-ті роки були розроблені системи безперервного розпізнавання усної мови з великим словниковим запасом (LVCSR). Ці системи могли розпізнавати тисячі слів і почали знаходити комерційне застосування. Вони розбивали усну мову на невеликі звукові одиниці, передбачаючи найбільш вірогідні слова та фрази з великого словникового запасу на основі цих звуків і використовували правила контексту та граматики для розуміння усного мовлення.

Початок 2000-х: системи розпізнавання усної почали включати вдосконалені статистичні мовні моделі та методи машинного навчання. Стандартом стала комбінація прихованих марківських моделей із моделями суміші Гауса. Моделі суміші Гауса — це спосіб представлення складних даних шляхом поєднання кількох простих дзвоноподібних (bell-shaped) кривих для моделювання різних шаблонів або особливостей у цих даних, допомагаючи розпізнавати такі речі, як звуки мови, фіксуючи їхню мінливість.

Середина 2000-х: завдяки потужнішим комп’ютерам і більшим наборам даних точність систем автоматичного розпізнавання усної мови значно підвищилася. Наявність даних розмовної мови (наприклад, у формі подкастів, відео) дозволила краще тренувати моделі.

У 2010-х відбулося впровадження глибинного навчання, зокрема глибоких нейронних мереж. Ця подія ознаменувала переломний момент для галузі [4]. Глибокі нейронні мережі могли моделювати складні шаблони в мовних даних більш ефективно, ніж традиційні системи прихованих марківських моделей зі змішаними моделями Гауса. Протягом цього десятиліття також спостерігалося зростання популярності наскрізних(end-to-end) моделей, як, наприклад, часова класифікація Connectionist (CTC) [39] і моделей на основі уваги, що зробили моделі розпізнавання усної мови більш доступними тим, що вони усунули потребу в проміжних кроках, таких як виділення ознак і зіставлення вхідних звуків/фонем з вихідними текстом [5]. Саме у цей час такі компанії, як Google, Apple, Amazon і Microsoft, зробили розпізнавання мовлення основною технологією для віртуальних помічників (наприклад, Google Assistant, Siri, Alexa).

Початок 2020-х років: тенденція до більших і потужніших моделей продовжилась із такими системами, як Whisper від OpenAI і Translatotron від Google. Ці моделі можуть працювати з кількома мовами, акцентами та навіть перекладом з усного в усне мовлення [37].

РОЗДІЛ 1. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

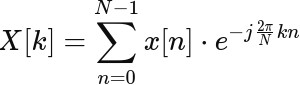
Як працює обробка природної мови (ОПМ) [12]? Моделі ОПМ знаходять зв’язки між складовими частинами мови — наприклад, літерами, словами та реченнями, знайденими в послідовному наборі даних. Архітектури ОПМ використовують різні методи для попередньої обробки даних (preprocessing), вилучення ознак (feature extraction) і моделювання [13].

**Попередня обробка даних**: часто перш ніж модель почне працювати з вхідною послідовністю, її потрібно попередньо обробити, щоб покращити продуктивність моделі або перетворити аудіо у формат, який модель може зрозуміти. У попередній обробці даних для систем розпізнавання усної мови використовують наступні методи:

1. Перетворення на рівномірну частоту дискретизації: важливо переконатися, що всі аудіозаписи дискретизуються з однаковою частотою, наприклад 16 кГц або 44,1 кГц, щоб підтримувати послідовність обробки даних.
2. Фільтрування: застосування фільтрів для видалення фонового шуму та покращення якості аудіосигналу та методи зменшення шуму: використання алгоритмів для зменшення шуму зі збереженням мовних сигналів.
3. Нормалізація гучності: регулювання амплітуди аудіозаписів, для забезпечення однакового рівня гучності, що допомагає підтримувати узгодженість різних записів.
4. Сегментація мовлення та не-мовлення: визначення та відокремлення сегментів аудіо, які містять мову, від тих, які її не містять. Це може включати виявлення тиші або не-мовних сегментів.

**Вилучення ознак** передбачає перетворення необроблених аудіосигналів у набір значеннєвих представлень (ознак), які можуть використовуватися моделями машинного навчання для розпізнавання та інтерпретації мовлення. Цей процес має вирішальне значення, оскільки необроблені аудіодані, як правило, мають велику кількість вимірів і містять багато зайвої та нерелевантної інформації. Мета виділення ознак полягає в тому, щоб відокремити суттєву інформацію, яка може зафіксувати характеристики мовного сигналу в більш компактній та дискримінаційній формі. До популярних технік вилучення ознак можна віднести:

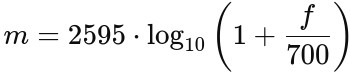
**Мел-частотні кепстральні коефіцієнти (МЧКК)**: Для аналізу сигнал спочатку розбивається на крихітні частини (так звані кадри – frames). Це дає змогу спостерігати, як звук змінюється в дуже малих сегментах часу. Спочатку застосовується швидке перетворення Фур’є (ШПФ) до кожного кадру аудіосигналу, щоб перетворити його з часової області в частотну. Формула виглядає наступним чином:



де X[k] – коефіцієнт Фур’є для частоти k x[n] – значення сигналу у час n

N – кількість точок у сигналі j — уявна одиниця.

Таким чином кожен із цих невеликих фрагментів аналізується, щоб побачити, які частоти присутні. Різні звуки в мові (наприклад, «а» або «е») мають різну частоту.

Далі, щоб імітувати сприйняття звуку людським вухом, частотний спектр, отриманий від ШПФ, пропускається через набір фільтрів Мел. Шкала Мел — це шкала сприйняття, яка більш чутлива до нижчих частот, що відповідає тому, як люди сприймають звук. Набір фільтрів складається з трикутних фільтрів, що перекриваються. Вони застосовуються до спектру ШПФ, що призводить до набору частотних компонентів Мел. Формула наступна:

де m — частота Мел

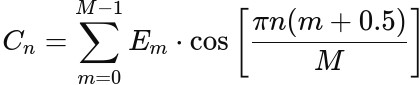
f — фактична частота в Гц.

Далі, діапазон стискається за допомогою логарифма. Це схоже на те, як людське вухо сприймає гучність. Формула виглядає так:



де Em — логарифм енергії м-го виходу, відфільтрованого Мел,

Sm — вихід м-го фільтра Мел.

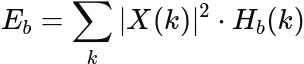
Нарешті, ми застосовуємо до цих стислих значень математичну операцію, яка називається дискретним косинусним перетворенням, яка дає нам набір коефіцієнтів (чисел). Ці коефіцієнти і є вилученими ознаками. Формула наступна:

де Cn — n-й Мел-частотний кепстральний коефіцієнт, M – загальна кількість фільтрів Мела,

Em — логарифм енергії m-го виходу, відфільтрованого Мел.

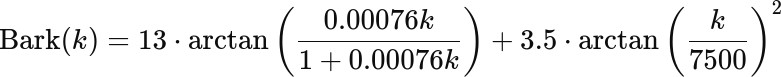
У контексті обробки сигналу та виділення ознак, «енергія» відноситься до міри потужності або потужності сигналу в межах певного діапазону частот.

Слід підмітити, що дискретне косинусне перетворення є опціональним. Значення вже готові до використання і без цієї операції. Така скорочена техніка вилучення ознак має назву **логарифмічна спектограма Мела** (саме її ми будемо використовувати для нашої моделі). Тут спектограма – це візуальне представлення спектру частот у сигналі, коли він змінюється з часом.

**Перцептивне лінійне передбачення**: Подібне до Мел-частотних кепстральних коефіцієнтів, головна відмінність полягає в тому, що цей підхід бере використання моделі слуху людини ще далі за допомогою банку фільтрів шкали Барка, кривої рівної гучності та компресії кубічного кореня. Як і в підході Мел-частотних кепстральних коефіцієнтів, сигнал спочатку розбивається на крихітні частини (так звані кадри – frames), та застосовується швидке перетворення Фур’є до кожного кадру аудіосигналу, щоб перетворити його з часової області в частотну. Формула вже була наведена вище. Далі, коефіцієнт Фур’є для частоти k пропускається через банк фільтрів шкали Барка. Шкала Барка – це спосіб вимірювання звукових частот, який імітує те, як люди сприймають висоту. Замість використання стандартної шкали Герца (Гц), яка вимірює частоту лінійно, шкала Барка розроблена для відображення нелінійного способу сприйняття звуків нашими вухами та мозком. Формула виглядає наступним чином:

де Eb — енергія в b-й смужці Барка,

Hb(k) — частотна характеристика b-го фільтра шкали Барка (виділяє певні частини звукового сигналу, зосереджуючись на певному діапазоні частот – смузі).

Шкалу Барка можна розрахувати за допомогою наступної формули:

де k – індекс частоти.

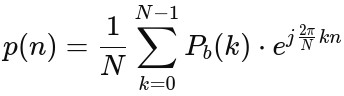
Енергія в кожній смузі Барка зважена рівною кривою гучності, яка моделює чутливість людського вуха до різних частот:

Eb′ — зважена енергія рівної гучності в bb-му діапазоні Барка,

Lb — крива рівної гучності на b-й смузі Барка (часто заснована на кривій Флетчера- Мансона).

Щоб наблизити нелінійне сприйняття гучності людським вухом, застосовується стиснення кубічного кореня:

де Pb — стиснута енергія b-ї смуги Барка.

Щоб перетворити стислі енергії назад у часову область, виконується зворотне дискретне перетворення Фур’є:

де p(n) представляє перцептивно змінений сигнал у часовій області.

На останок, використовується кодування з лінійним предиктором (LPC), що полягає у перетворенні аналогового сигналу в цифрову форму, і використовується в основному для представлення спектральної огинаючої цифрового сигналу мовлення в стислому вигляді, з використанням інформації лінійної прогностичної моделі.

На виході отримаємо коефіцієнти LPC, які можна використовувати в якості вилучених ознак.

**Моделювання**: після попередньої обробки дані вводяться в архітектуру моделі автоматичного розпізнавання усного мовлення, яка моделює їх для виконання різноманітних завдань. Вилучені ознаки, отримані за допомогою методів описаних вище, можут бути використані як вхідні ознаки для різних моделей, таких як приховані моделі Маркова або глибокі нейронні мережі. Коефіцієнти (вилучених ознак) часто піддаються нормалізації, щоб забезпечити послідовне масштабування та покращити навчання моделі. Методи нормалізації включають нормалізацію z- показника або нормалізацію довжини голосового тракту.

Нормалізація Z-оцінки, також відома як стандартизована нормалізація оцінки, — це техніка, яка використовується в контексті автоматичного розпізнавання усної мови для стандартизації ознак, отриманих із мовного сигналу. Основна мета нормалізації z-показника полягає в тому, щоб зробити ознаки більш зручними для порівняння, методом центрування їх навколо середнього значення нуля та масштабуючи їх, щоб отримати стандартне відхилення одиниці. Формула має наступний вигляд:

де x – оригінальне значення ознаки μ – середнє значення ознаки

σ — стандартне відхилення ознаки.

z — нормалізоване значення ознаки (z-показник).

Нормалізація довжини голосового тракту — це техніка, яка використовується для компенсації відмінностей у довжині голосового тракту між різними мовцями. Оскільки довжина голосового тракту впливає на резонансні частоти мови, нормалізація регулює частотну вісь мовного сигналу, щоб зменшити мінливість, спричинену цими анатомічними відмінностями. Нормалізація полягає у застосуванні функції нелінійної деформації частоти до мовного сигналу. Ця функція відображає вихідну вісь частот на нову вісь, яка компенсує різницю в довжині голосового тракту. Коефіцієнт деформації зазвичай визначається під час фази навчання, де оцінюється найкращий коефіцієнт деформації для кожного промовця.

Мета мовної моделі полягає у тому, щоб прослухати фрагмент аудіо, та згенерувати на основі почутого транскрипцію. Нині для завдання автоматичного розпізнання усної мови є актуальними дві категорії моделей: традиційні методи машинного навчання та методи глибинного навчання.

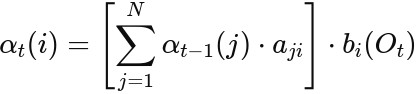
До традиційних методів машинного навчання належать: приховані моделі Маркова, моделі суміші Гауса.

До методів глибинного навчання належать: рекурентні нейронні мережі, довга короткочасна пам’ять, моделі “послідовність-до-послідовності” та трансформер.

**Приховані моделі Маркова (ПМП)** були основою традиційних систем автоматичного розпізнавання усної мови [1]. Вони моделюють послідовність звуків мови (фонем) як серію станів, де кожен стан представляє сегмент мовного сигналу [6]. Модель складається з:

* Станів S,
* Спостережень O (ці спостереження можуть бути ознаками, як, наприклад, кепстральні коефіцієнти Мел-частоти).
* Ймовірностей переходу. Ймовірність переходу з одного стану в інший задається матрицею ймовірностей переходу A. Елемент a[i][j] представляє ймовірність переходу зі стану S[i] до стану S[j]:
* Ймовірності викиду. Кожен стан S[i] публікує спостереження відповідно до розподілу ймовірностей. Імовірність викиду b[i](O[t]) — це ймовірність спостереження O[t] у заданому стані S[i]:
* Ймовірності початкового стану. Початковий розподіл ймовірностей π[i] представляє ймовірність запуску моделі у стані S[i]:

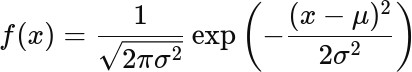
Для завдання автоматичного розпізнавання усної мови використовують такі алгоритми прихованих моделей Маркова, як прямий алгоритм, алгоритм Вітербі [3], алгоритм Баума — Велша [2].

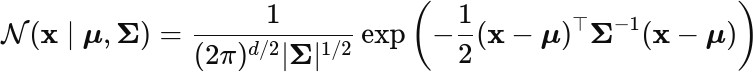
Прямий алгоритм використовується для обчислення ймовірності спостереження певної послідовності акустичних характеристик (наприклад, МЧКК) з урахуванням ПМП. Ця ймовірність допомагає визначити, наскільки добре конкретна ПМП (що відповідає слову або фонемі) відповідає мовному сигналу, що спостерігається:

Алгоритм Вітербі грає важливу роль для автоматичного розпізнавання усної мови, адже він знаходить найбільш ймовірну послідовність прихованих станів (фонем або слів), які могли б створити послідовність акустичних характеристик, що спостерігається. Зазвичай використовується для визначення найбільш ймовірної транскрипції вимовленого речення.

Алгоритм Баума-Велча використовується на етапі навчання системи. Це дозволяє оцінити параметри прихованої моделі маркова (ймовірності переходу A, ймовірності викиду B та початкові ймовірності π) на основі набору навчальних даних (усні висловлювання та їх транскрипції). Алгоритм передбачає ітераційне оновлення параметрів для максимізації ймовірності спостережуваних даних. Він не має єдиної стислої формули, але передбачає обчислення очікуваної кількості переходів і викидів, а потім нормалізацію цих підрахунків для оновлення параметрів моделі.

**Модель суміші Гауса (МСГ) [7]** є імовірнісною моделлю для представлення нормально розподілених субпопуляцій у загальній сукупності. Змішані моделі загалом не вимагають знати, до якої субпопуляції належить точка даних, що дозволяє моделі вивчати субпопуляції автоматично. Оскільки призначення субпопуляції невідоме, це є формою неконтрольованого навчання. У автоматичному розпізнаванні усної мови мета полягає в тому, щоб зіставити звуковий сигнал із відповідними фонетичними одиницями, які є будівельними блоками слів. Проте мовні сигнали дуже різноманітні через такі фактори, як відмінності мовців, фоновий шум і варіації вимови. Моделі суміші Гауса добре підходять для фіксації цієї мінливості шляхом моделювання розподілу акустичних характеристик для кожної фонетичної одиниці. Ці моделі часто використовують у поєднанні з прихованими моделями Маркова. Кожна фонетична одиниця (або стан у прихованій моделі Маркова) представлена за допомогою моделі суміші Гауса, яка моделює розподіл ймовірностей цих векторів ознак.

 Модель суміші Гауса можна розуміти як комбінацію кількох дзвоноподібних кривих (розподілу Гауса), які разом описують більш складний набір даних. Модель складається з кількох компонентів Гауса, кожен з яких представляє різну субпопуляцію в акустичному просторі фонетичної одиниці. Наприклад, така фонема, як /a/, може мати кілька компонентів Гауса, кожен з яких відповідає різним артикуляційним або акустичним варіаціям фонеми. Загальна функція щільності ймовірності для фонетичної одиниці є зваженою сумою цих компонентів Гауса. Ваги відображають ймовірність того, що вектор ознак буде створений кожним компонентом. В одному вимірі функція щільності ймовірності розподілу Гауса визначається так:

Для d-вимірного вектора значень x функція матиме наступний вигляд:

де μ: середній вектор розподілу Гауса.

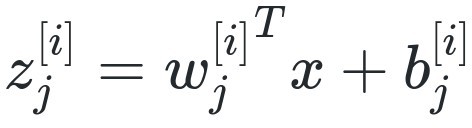
Σ: коваріаційна матриця розподілу Гауса.

Параметри моделі суміші Гауса (середні значення, коваріації та ваги суміші) зазвичай оцінюються за допомогою алгоритму очікування-максимізації. Алгоритм скаладається з двох кроків: очікування та максимізації. На кроці очікування робиться припущення щодо відсутні даних. За допомогою поточних параметрів моделі оцінюється до якої групи, швидше за все, належить кожна точка даних. На кроці максимізації модель оновлює значення параметрів, щоб краще відповідати даним, на основі попередніх припущень. Весь ітераційний процес повторюється, допоки алгоритм не зійдеться, даючи оцінку максимальної правдоподібності [10].

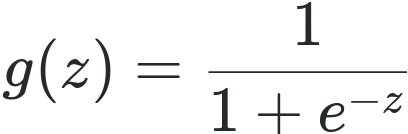
Моделі суміші Гауса можуть адаптуватися до нових даних, що робить їх ефективними для методів адаптації мовця, коли модель точно налаштовується на голос конкретного мовця, покращуючи точність розпізнавання.

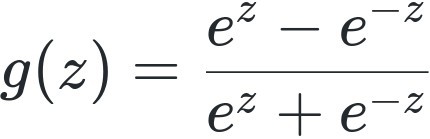
Нейронні мережі складаються з нейронів [8]. Кожний нейрон на вхід отримує вектор x з розміром n та вираховує на вихід скалярне значення y^, яке ще називають спрогнозоване значення. Кожний вектор x cкладається зі значень усіх особливостей (features) одного із екземплярів датасету. Для автоматичного розпізнавання усної мови екземпляром є аудіофайл.

Для кожного значення з x нейрон має відповідний параметр, що називається вагою (weight). У додаток до цього, кожний нейрон ще має вільний параметр, що називається ухил (bias). Ваги та ухил нейрона змінюються під час тренування нейронної мережі. На кожній ітерації тренування нейрон вираховує зважене середнє число зі значень вектора x за допомогою ваг, та додає до отриманого значення ухил.

Узагальнена формула обчислення вихідного значення нейрона до активаційної функції:

Після цього результат передається у активаційну функцію g, яка зазвичай нелінійна. Найчастіше в якості активаційних функцій використовують:

Sigmoid:



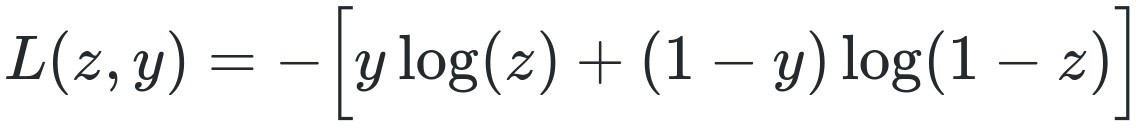
Tanh:

ReLU: 

Активаційна функція – один iз ключових елементів нейронної мережі. Без них нейронна мережа стає комбінацією лінійних функцій, що у свою чергу також є лінійною функцією. На даний момент найпопулярнішою активаційною функцією для прихованих шарів є ReLu.

Sigmoid часто використовують у вихідному шарі коли мають справу з бінарною класифікацією і потрібно, щоб вихідний шар видавав результат у діапазоні від 0 до 1, що можна інтерпретувати як ймовірність належності до позитивного класу. Якщо результат більше або дорівнює 0.5, він класифікується як належний до позитивного класу; інакше він класифікується як належний до негативного класу. Для оптимізації розрахунків важливо використовувати матриці для ваг W та вектор для ухилу B. Це дає змогу проводити розрахунок не по одному нейрону за раз, а на цілому шарі (layer) нейронів одразу. Це називається векторизацією.

Для тренування нейронна мережа потребує великої кількості даних, часто мільйонів екземплярів. Тренувати нейронну мережу на одному екземплярі за раз не ефективно з точки зору часу та ресурсів. Тому для пришвидшення процесу із екземплярів формують партії (batches), які зазвичай містять 2^n екземплярів (m = 16, 32, 64, 128, …). Для того аби провести розрахунки шару для всієї партії одразу, необхідно ще раз застосувати векторизацію: замість вектора a, використовувати матрицю A, яка складається з a для m екземплярів партії.

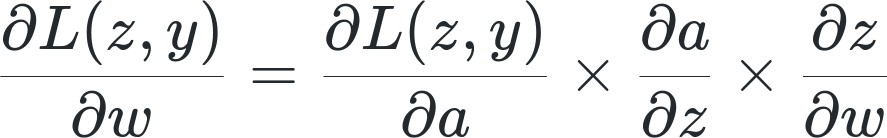
Прогрес тренування у нейронній мережі вимірюють за допомогою функції втрат (loss function). Ціль функції втрат – показати наскільки відрізняється поточна точність нейронної мережи від ідеалу. Функція втрат, що використовується для завдань бінарної класифікації називається бінарна перехресна ентропія (binary cross-entropy). Її формула:

Значення цієї функції повинно зменшуватися з часом, а точність нейронної мережі зростати.

Навчання нейронної мережі полягає у зміненні ваг W та ухилу b таким чином, аби мінімізувати функцію втрат. Для цього використовується алгоритм під назвою метод зворотнього поширення (backpropagation). Алгоритм працює так, що спочатку вхідні дані подаються в мережу, яка створює вихід. Цей вихід порівнюється з бажаним результатом, а різниця між ними обчислюється як помилка. Потім помилка поширюється мережею у зворотному напрямку, шар за шаром.

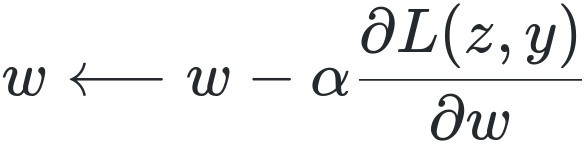
Під час зворотного поширення помилка використовується для коригування ваг зв’язків між нейронами. Вагові коефіцієнти коригуються пропорційно помилці та похідній функції активації нейронів. Ця похідна вимірює, наскільки змінюється вихід нейрона щодо його входу.

Вагові коефіцієнти коригуються за допомогою алгоритму оптимізації, такого як стохастичний градієнтний спуск (stochastic gradient descent). Цей алгоритм ітеративно оновлює вагові коефіцієнти в напрямку, що мінімізує помилку, доки помилка не досягне мінімуму або не буде досягнуто бажаного рівня точності.

dW та db, що вираховуються за допомогою правила ланцюга (chain rule), це часткові похідні по W та b відповідно. Виміри dW та db такі самі як і у W та b.

Процес зворотного поширення повторюється протягом багатьох ітерацій, поки мережа не досягне задовільного рівня продуктивності.

Ваги коректуються за допомогою наступної формули:



де α – швидкість навчання (learning rate) – гіперпараметр, що допомагає контролювати швидкість зміни параметрів.

Ухил вираховується за аналогічною формулою.

Гіперпараметр – це характеристика, значення якої розробник нейронної мережі повинен задати вручну. Приклади: швидкість навчання, активаційна функція, кількість нейронів у шарі. Параметр – це характеристика, значення якої нейронна мережа знаходить власноруч. Розробник нейронної мережі тільки ініціює значення за потреби. Приклади: ваги, ухил.

Вибирати швидкість навчання потрібно дуже прискіпливо, адже якщо значення замале – нейронна мережа буде вчитися занадто довго, якщо ж значення завелике – не вдасться досягнути мінімуму.

Важливо уникати перенавчення (overfitting) нейронної мережі. Перенавчення є поширеною проблемою в машинному навчанні. Перенавчена модель підлаштовується під тренувальні дані надто точно, аж до того, що починає фіксувати шум і випадкові коливання в даних замість базових шаблонів. Це може спричинити погану роботу моделі на нових, невидимих раніше даних, оскільки вона, по суті, запам’ятала тренувальні дані, а не вивчила базові зв’язки між вхідними функціями та вихідною змінною. Перенавчення може статися, коли модель надто складна або має забагато параметрів відносно обсягу тренувальних даних. Це означає, що модель має надто велику гнучкість і може дуже точно відповідати навчальним даним, але може погано узагальнювати нові дані.

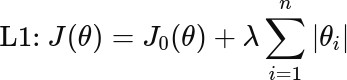
Для тренування моделі не слід задіювати усі дані. Загальний набір даних потрібно розділити на 3 нерівні частини: для тренування, валідації та тесту (train/val/test split). Відсоток розподілу залежить від кількості даних, але частина для тренування завжди найбільша, без винятків. Якщо датасет порівняно невеликий, то розподіл може бути приблизно таким: 70/20/10 (%). Якщо даних дуже багато (мільйони екземплярів), то можна виділити більший відсоток під тренування: 98/1/1 (%).

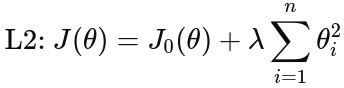
Валідаційна частина використовується під час тренування для того, аби перевірити наскільки добре модель узагальнює вивчене на нових даних.

Тренування моделі відбувається у циклах. Цикл тренування завершено, коли всі дані тренувальної частини були задіяні у партіях. Для досягнення кращих результатів цикл потрібно повторювати багато разів (10+). Валідаційна частина використовується під час кожного циклу тренування (epoch). Оцінка моделі на валідаційній частині напряму впливає на рішення під час розробки моделі, тому важливо додатково перевіряти проти перенавчення на валідаційній частині. Саме для цього після тренування використовують тестову частину.

Один із способів виявити перенавчення — порівняти продуктивність моделі за даними тренування та даними валідації. Якщо точність моделі на даних валідації починає знижуватися, навіть якщо точність на даних тренування продовжує покращуватися, це може бути ознакою перенавчення. Для запобігання перенавчення може знадобитися спростити модель, зменшити кількість параметрів або використати методи регуляризації.

Регуляризація – це один із основних методів зменшення перенавчення. Він полягає у додаванні нового елементу до функції втрат [23]. Призначення цього елементу - штрафувати параметри (ваги), значення яких занадто велике. Дві найпопулярніші види регуляризації – L1 та L2.

У L1 штраф пропорційний до модульного значення параметрів, у L2 штраф пропорційний до квадратів значень параметрів. Формули L1, L2:

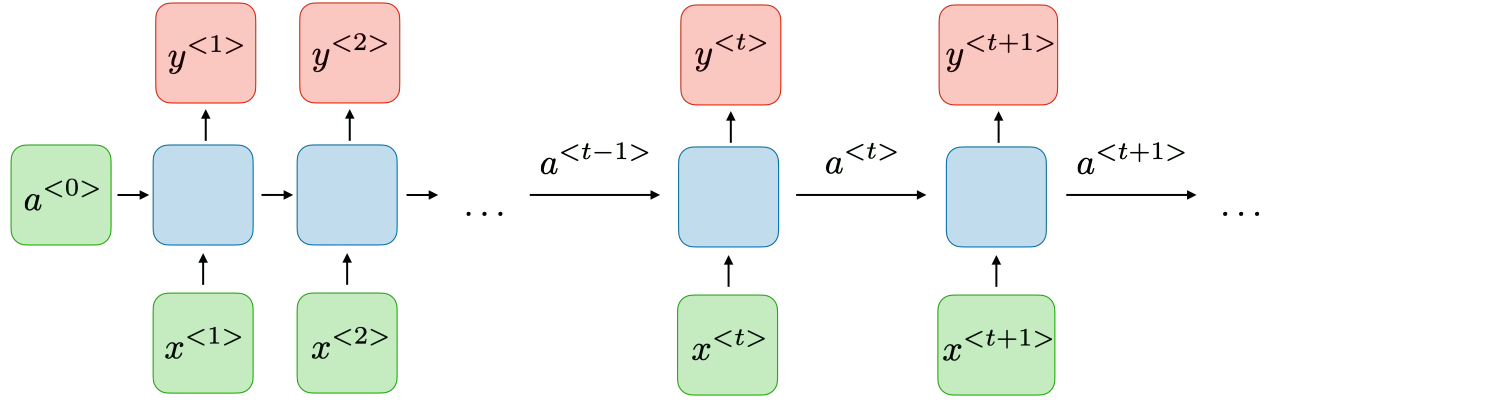


L1 більш бажана, адже вона знаходить неважливі особливості у даних, та ігнорує їх прирівнюючи відповідні параметри до нуля. До того ж, L2 не дає хороших результатів коли дані містять багато винятків. L2 використовує значення квадратів параметрів, що призводить до мінімізації впливу винятків за рахунок більш популярних екземплярів. Інший розповсюджений вид регуляризації – метод виключення (dropout). Він полягає у наступному: кожний нейрон, за виключенням вихідного шару (output layer), має шанс p (зазвичай 0.2 – 0.5) бути проігнорованим (прирівняним до 0) для наступної партії у циклі тренування.

Таким чином, ми тренуємо меньш складну модель, яку ми знеохочуємо залежати від конкретних нейронів і змушуємо шукати більш узагальнені особливості. В результаті модель стає стійкішою до нових даних. Завелика кількість циклів тренування також призводить до перенавчання. У такому разі використовують техніку під назвою рання зупинка (early stopping). Тренування завершується коли результат поточного циклу гірший за попередній.

Ще одним важливим методом боротьби з перенавченням є нарощення даних (data augmentation). Метод полягає у штучному створенні нових екземплярів на основі існуючих, використовуючи математичні перетворення. Приклади перетворень для зображень: обертання на 90 градусів, зеркальне перевертання, зміна яскравості, зміна контрасту, обрізання тощо. Цей підхід збільшує розмір набору даних та його різноманітність. Це покращує можливість узагальнювати дані та бути стійкішим до нових, раніше небачених екземплярів. Нарощення даних є особливо важливим коли набір даних відносно невеликого розміру.

**Рекурентні нейронні мережі** — це клас штучних нейронних мереж, у якому з'єднання між вузлами утворюють граф орієнтований у часі [11]. Це створює внутрішній стан мережі, що дозволяє їй проявляти динамічну поведінку в часі. На відміну від нейронних мереж прямого поширення, РНМ можуть використовувати свою внутрішню пам'ять для обробки довільних вхідних послідовностей. Це робить їх здатними до таких задач, як розпізнавання несегментованого неперервного рукописного тексту та розпізнавання мовлення. Архітектура РНМ виглядає наступним чином [9]:

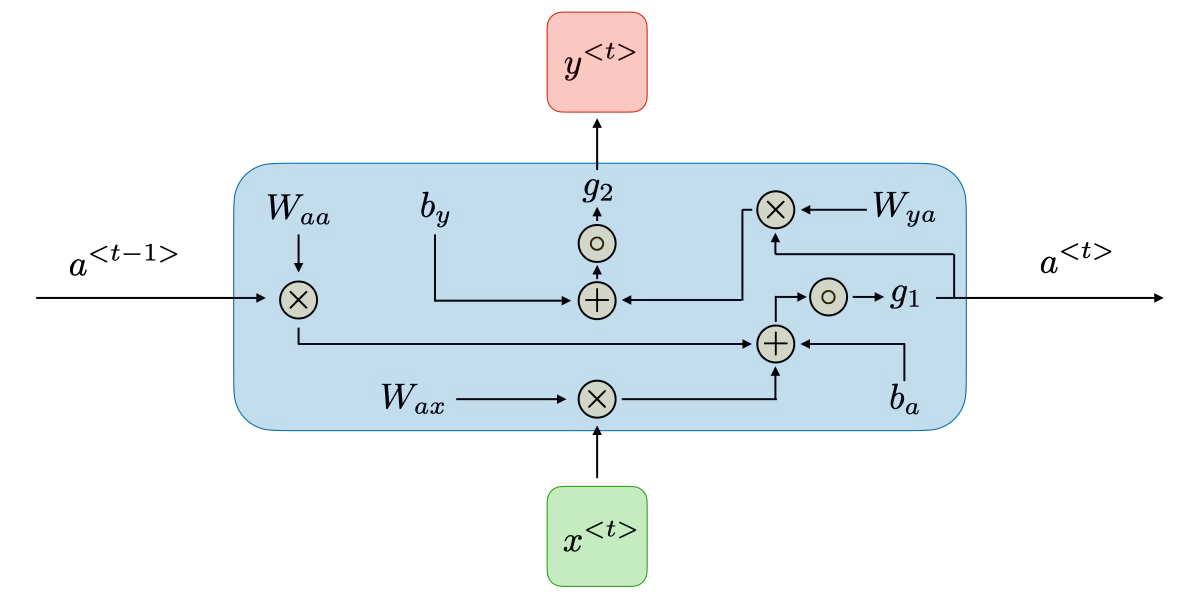


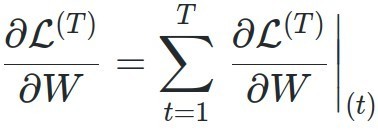
Для кожного тимчасового кроку t активація a<t> і вихід y<t> виражаються таким чином:

де: Wax, Waa, Wya, ba, by — це часові коефіцієнти (ваги та ухил), що повторно використовуються.

g1, g2 - функції активації (зазвичай sigmoid, tanh, ReLU).

Більш детальна ілюстрація того, що відбувається на кожного кроці часу t:



Функція втрат у машинному навчанні вимірює, наскільки прогнози моделі відповідають фактичним результатам. Вона кількісно визначає похибку між прогнозованими та справжніми значеннями, керуючи процесом навчання моделі, щоб мінімізувати цю похибку. Оптимізуючи функцію втрат, модель покращує свою продуктивність і точність. У випадку рекурентної нейронної мережі функція втрат L усіх часових кроків визначається на основі втрат на кожному часовому кроці t. Для рекурентних нейронних мереж зворотнє поширення здійснюється в кожен момент часу. На кроці часу T похідна втрати L відносно вагової матриці W виражається таким чином:

Переваги рекурентних нейронних мереж:

* Можливість обробки вхідних даних будь-якої довжини.
* Розмір моделі не збільшується разом із розміром вхідних даних.
* Розрахунок враховує історичну інформацію.
* Ваги повторно використовуються на кожному кроці часу t.

Недоліки рекурентних нейронних мереж:

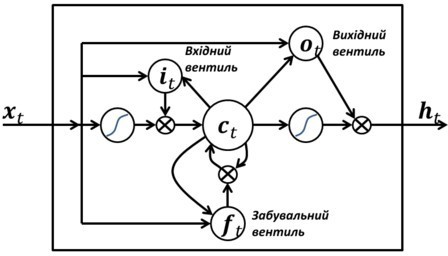
* Повільне обчислення.
* Труднощі з доступом до давньої інформації.

Явища зникнення/вибуху градієнта (vanishing/exploding gradient) часто зустрічаються в контексті РНМ. Причина, полягає в тому, що РНМ важко охопити довгострокові залежності через мультиплікативний градієнт, який може експоненціально зменшуватися/збільшуватися щодо кількості шарів.

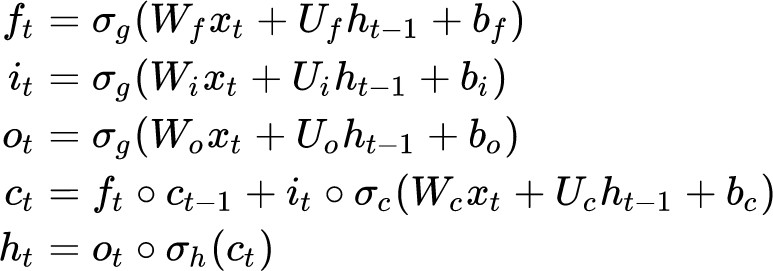
* Неможливо розглянути будь-який майбутній вхід для поточного стану (двонаправлені РНМ вирішують цю проблему).

**Довга короткочасна пам'ять** (ДКЧП, long short-term memory, LSTM, [14]) — це тип рекурентної нейронної мережі (РНМ), спрямованої на вирішення проблеми зникнення/вибуху градієнта, присутньої в традиційних РНМ. Його відносна нечутливість до довжини розриву є його перевагою перед РНМ. ДКЧП має на меті забезпечити короткочасну пам’ять для РНМ, яка може тривати тисячі кроків, тобто «довгу короткочасну пам’ять». ДКЧП застосовують для класифікації, обробки та прогнозування даних на основі часових рядів, наприклад у рукописному тексті, розпізнаванні мовлення, машинному перекладі, виявленні мовленнєвої активності, керуванні роботами, відеоіграх та охороні здоров’я [15].

Звичайний блок ДКЧП складається з клітини, вхідного вентиля, вихідного вентиля та забувального вентиля. Клітина запам'ятовує значення протягом довільних проміжків часу, а три вентиля регулюють потік інформації в клітину та з неї. Забувальний вентиль вирішує, яку інформацію відкинути з попереднього стану, призначаючи попередньому стану порівняно з поточним станом значення від 0 до 1. (Округлене) значення 1 означає збереження інформації, а значення 0 означає відкидання інформації. Вхідний вентиль вирішує, які частини нової інформації зберігати в поточному стані, використовуючи ту саму систему, що й забувальний вентиль [16].

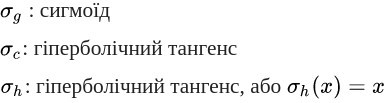
Вихідний вентиль контролює, які фрагменти інформації в поточному стані виводити, призначаючи значення від 0 до 1 інформації, враховуючи попередній і поточний стани. Вибіркове виведення відповідної інформації з поточного стану дозволяє мережі ДКЧП підтримувати корисні довгострокові залежності для прогнозування як у поточному, так і в майбутньому часових кроках. Блок ДКЧП можна візуалізувати наступним чином:

Формули для ДКЧП виглядають наступним чином:



де

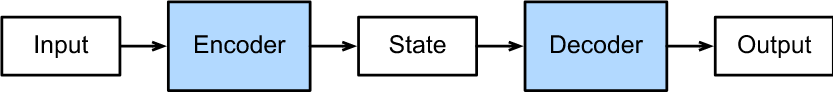
де активаційні функції:



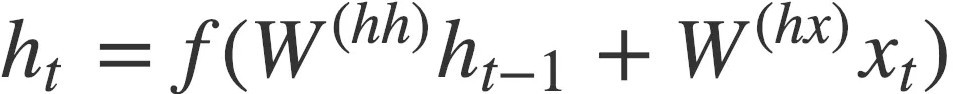
Довга короткочасна пам’ять (ДКЧП) може вирішувати численні завдання, які неможливо вирішити попередніми алгоритмами навчання рекурентних нейронних мереж (РНМ). Мережі ДКЧП є потужними для завдань, що включають послідовності даних із ієрархічною структурою, і вони можуть автоматично вивчати та використовувати цю структуру без попереднього знання конкретної декомпозиції [17, 18, 19, 20, 21, 25].

**Модель «послідовність до послідовності» (Seq2Seq)**. Вперше представлена компанією Google у 2014 році модель seq2seq зіставляє вхідні дані фіксованої довжини з вихідними даними фіксованої довжини, де довжина вхідних і вихідних даних може відрізнятися [24].

Наприклад, переклад «Що ти сьогодні робиш?» з української на китайську має 4 вхідних слова і виводить 7 символів ( 今天你在做什麼？ ) [29]. Звичайна ДКЧП або РНМ не може відобразити кожне слово з українського речення в китайське. Модель seq2seq використовується для вирішення подібних проблем [26].

Модель seq2seq складається з кодера, вектора кодера (state) та декодера [27]. Кодер фіксує контекст вхідної послідовності та надсилає його в декодер, який потім створює остаточну вихідну послідовність. Узагальнену архітектуру seq2seq можна зобразити таким чином:

Кодер зазвичай складається з декількох рекурентних блоків ДКЧП, де кожен блок приймає окремий елемент вхідної послідовності, збирає інформацію для цього елемента та поширює її до наступного блоку. Для завдання машинного перекладу, наприклад, вхідна послідовність дорівнює набору слів. Кожне слово позначане як Кожне слово позначане як x[i], де i – порядковий номер слова.

Прихований стан h[t] вираховується за допомогою наступної формули:

Вектор кодера — це остаточний прихований стан, отриманий з частини моделі кодера. Він розраховується за наведеною раніше формулою. Цей вектор має на меті інкапсулювати інформацію для всіх вхідних елементів, щоб допомогти декодеру робити точні прогнози. Він діє як початковий прихований стан частини моделі декодера.

Декодер складається з декількох рекурентних блоків ДКЧП, де кожен блок передбачає вихідний стан y[t] на кроці часу t. Кожний рекурентний блок приймає прихований стан від попереднього блоку та створює вихідний стан, а також свій власний прихований стан.

Для завдання машинного перекладу, наприклад, вихідна послідовність дорівнює перекладу. Кожне слово позначане як y[i], де i – порядковий номер слова. Будь-який прихований стан h[i] обчислюється за формулою:

Вихідний стан y[t] на кроці часу t обчислюється за допомогою прихованого стану на поточному кроці часу разом із відповідною вагою W(S). Softmax використовується для створення вектора ймовірності, який допомагає визначити остаточний результат (наприклад, перекладане слово в задачі машинного перекладу).

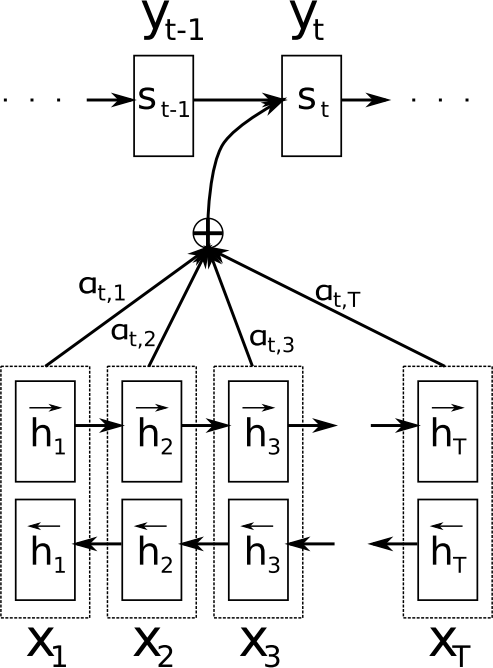
Формула вихідного стану виглядає наступним чином:

Потужність цієї моделі полягає в тому, що вона може відображати послідовності різної довжини: входи та виходи не корельовані, і їх довжина може відрізнятися. Це відкриває цілий ряд задач, які тепер можна вирішити за допомогою такої архітектури.

У контексті seq2seq моделі також важливо згадати про механізм уваги (attention) [33]. В моделі seq2seq, ми відображаємо вхідну послідовність у векторному представленні фіксованої довжини (вектор кодера), а потім вихідний стан на основі цього вектора. Можна помітити, що декодер використовує лише останній стан кодера – вектор кодера. Вектор кодера має передавати декодеру повний зміст вхідної послідовності. Це важко зробити, якщо вхідна послідовність має велику довжину [28].

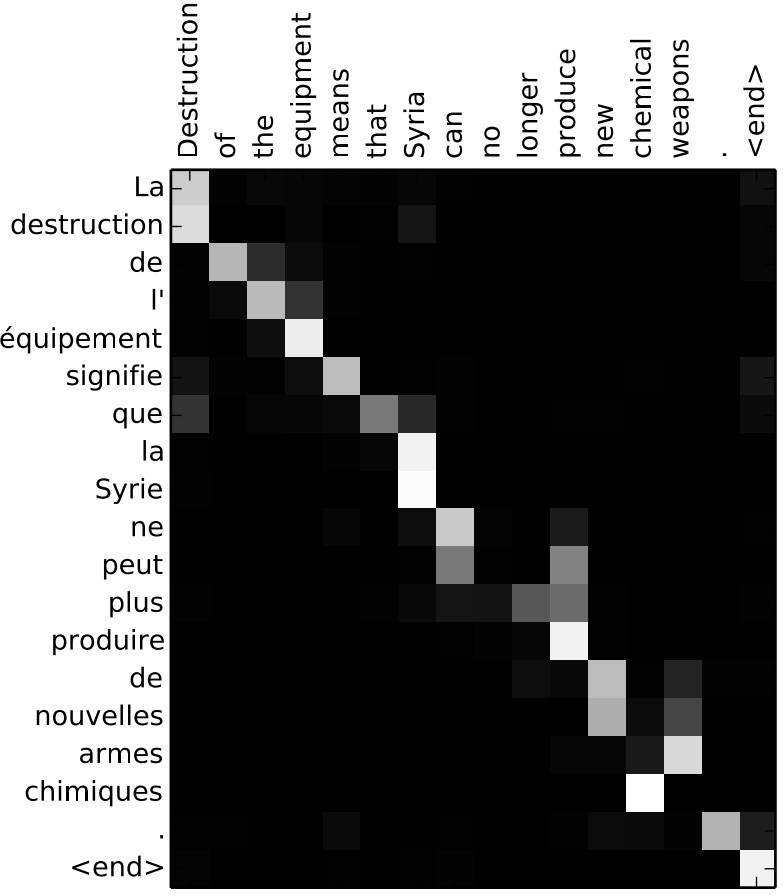
Наприклад, нехай речення на переклад містить 50 слів. Перше слово вхідної послідовності, ймовірно, корелює з першим словом вихідної. Але це означає, що декодер повинен враховувати інформацію 50 кроків тому, і ця інформація має бути якимось чином закодована у векторі кодера. Відомо, що рекурентні нейронні мережі мають проблеми з такими довгостроковими залежностями (зникнення градієнта).

Теоретично такі архітектури, як ДКЧП, повинні впоратися з цим, але на практиці довготривалі залежності все ще проблематичні [32]. За допомогою механізму уваги, модель більше не намагатиметься закодувати повну вихідну послідовність у вектор фіксованої довжини. Натомість, ми дозволяємо декодеру «переглядати» різні частини вхідної послідовності на кожному кроці генерації вихідного стану. Важливо те, що ми дозволяємо моделі дізнатися, на що звернути увагу, на основі вхідної послідовності та того, що вона сгенерувала до цього часу. Кожний стан вихідної послідовності y[t] на кроці часу t тепер залежить від зваженої комбінації всіх вхідних станів, а не лише останнього стану. а — це ваги механізму уваги, які зважують, скільки кожного вхідного стану слід враховувати для кожного вихідного стану. Якщо a[3,2] є великим числом, це означатиме, що декодер приділяє багато уваги другому стану у вхідній послідовності при створенні третього стану вихідної послідовності. Ваги a зазвичай нормалізуються до суми 1, щоб бути розподілом за вхідними станами (за допомогою функції softmax).

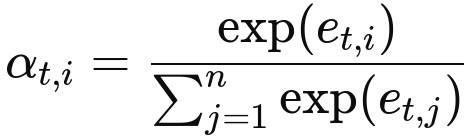
Візуально механізм уваги можна відобразити таким чином: 

Великою перевагою механізму уваги є те, що він дає можливість інтерпретувати та візуалізувати те, що робить модель. Наприклад, візуалізуючи матрицю ваги механізму уваги a під час перекладу речення, ми можемо зрозуміти, як модель виконує переклад.

З наступної ілюстрації буде видно, що під час перекладу з французької на англійську мову модель послідовно звертає увагу на кожний вхідний стан, але іноді вона звертає увагу на два слова одночасно, створюючи вихідний стан, як, наприклад, у перекладі «la Syrie» на «Syria».



На кожному часовому кроці t декодера ваги механізму уваги a[t,i] обчислюються для кожного прихованого стану h[i] кодера:

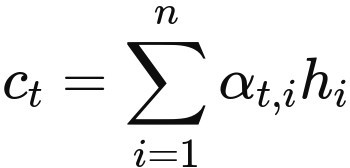


де:

e[t,i] — оцінка вирівнювання (alignment score), яка вимірює, наскільки вхідний стан навколо позиції i та вихідний стан у позиції t збігаються.

Оцінку можна обчислити за допомогою різних методів, наприклад скалярного добутку, конкатенації або нейронної мережі прямого поширення.

В звичайній seq2seq моделі вектор кодера дорівнює остаточному прихованому стану кодера. З використанням механізму уваги вектор кодера на кроці часу t є зваженою сумою усіх прихованих станів кодера:



Нарешті, вектор контексту c[t] поєднується з прихованим станом декодера h[t], щоб створити вихідний стан y[t] на кроці часу t:

Де: W\_c – вагова матриця.

[c\_t, h\_t] – конкатенація.

b\_c – ухил.

Softmax – ця функція гарантує, що сума ймовірностей вихідних даних дорівнює 1.

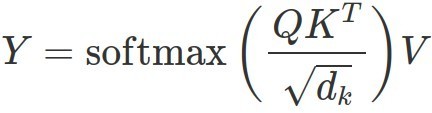
**Трансформер [35]**. Архітектура моделі, вперше описана в статті «Увага — це все, що вам потрібно» (Vasmani et al., “Attention is all you need”) [42] у 2017 році, уникає рекурентності і натомість повністю покладається на механізм самоуваги (self-attention) [34] для встановлення глобальних залежностей між вхідною та вихідною послідовністями . Оскільки цей механізм обробляє всі слова одночасно (замість одного за раз), це пришвидшує навчання та знижує вартість прогнозування порівняно з РНМ, особливо тому, що трансформер може використовувати паралелізацію. За останні роки архітектура трансформера революціонувала обробку природних мов [36].

Роль самоуваги, також відомої як масштабована скалярнодобуткова увага (scaled dot-product attention), полягає в тому, щоб наповнити модель контекстуальною обізнаністю, що, в свою чергу, дозволить моделі розрізняти значення окремих елементів у послідовності та динамічно регулювати їхній вплив на кінцевий результат. Механізм самоуваги утворює будівельні блоки трансформера. Сам механізм складається з наступних елементів: запитів Q (queries), ключів K (keys), значень V (values).

Запити Q – це елементи які шукають інформацію. Для кожного слова у вхідній послідовності обчислюється вектор запитів. Ці запити представляють те, на що моделі потрібно звернути увагу в межах послідовності.

Ключі K допомагають знайти важливі елементи послідовності. Подібно до запитів, вектори ключів обчислюються для кожного слова. Значення V несуть інформацію. Подібно до запитів та ключів, для кожного слова обчислюється вектор значень.

Після підготовки Q, K, V матриць слід обчислити оцінку уваги (attention score) для кожної пари слів вхідної послідовності. Оцінки уваги використовуються як ваги для виконання зваженого агрегування (weighted aggregation) векторів значень. Результат агрегування є результатом механізму самоуваги, що представляє розширене та контекстно обізнане представлення вхідної послідовності.

Математично механізм самоуваги можна представити наступним чином:

де:

— вимір векторів ключів, часто вибирається відповідно до виміру

векторів запитів та значень.

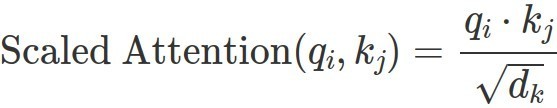
— це матриця скалярних добутків між векторами запитів та ключів.

Отриманий результат (матриця) використовується для зважування векторів значень.

Розіб’ємо формулу Y на декілька частин:

Оцінка уваги між вектором запитів q[i] та вектором ключів k[j] вираховується як скалярний добуток:

Для того, аби стабілізувати тренування та контролювати величину градієнту, скалярний добуток зменшується на коефіціент :



Зменшені оцінки уваги передаються в функцію softmax, щоб отримати вагові коефіцієнти уваги, сума яких дорівнює 1:



Нарешті, ваги механізму уваги використовуються для обчислення зваженої суми векторів значень:

На практиці, механізм самоуваги часто розширюється до механізму багатоголової уваги (multi-head attention). Замість того, щоб покладатися на один набір вивчених трансформацій Wq, Wk, Wv, багатоголова увага використовує кілька таких наборів трансформацій, або «голов». Кожна голова зосереджена на різних аспектах або зв’язках у вхідній послідовності. Результати цих голов об’єднуються для отримання остаточного результату самоуваги. Механізм багатоголової уваги дозволяє моделям отримувати різні типи інформації одночасно. Формула для багатоголової уваги виглядатиме так:



де:

— матриця проекції, що перетворює об’єднані результати кількох голов уваги в єдине уніфіковане представлення з бажаною розмірністю.

h — кількість голів.

Одним із критичних аспектів механізму самоуваги є те, що він за своєю суттю не фіксує порядок елементів у вхідній послідовності, оскільки обчислює увагу лише на основі вмісту. Щоб усунути це обмеження, до вхідної послідовності в трансформері додаються позиційні кодування. Ці кодування забезпечують модель інформацією про позиції слів у послідовності, що дозволяє розрізняти слова з однаковим змістом, але різними позиціями. Формули для позиційного кодування виглядають наступним чином:

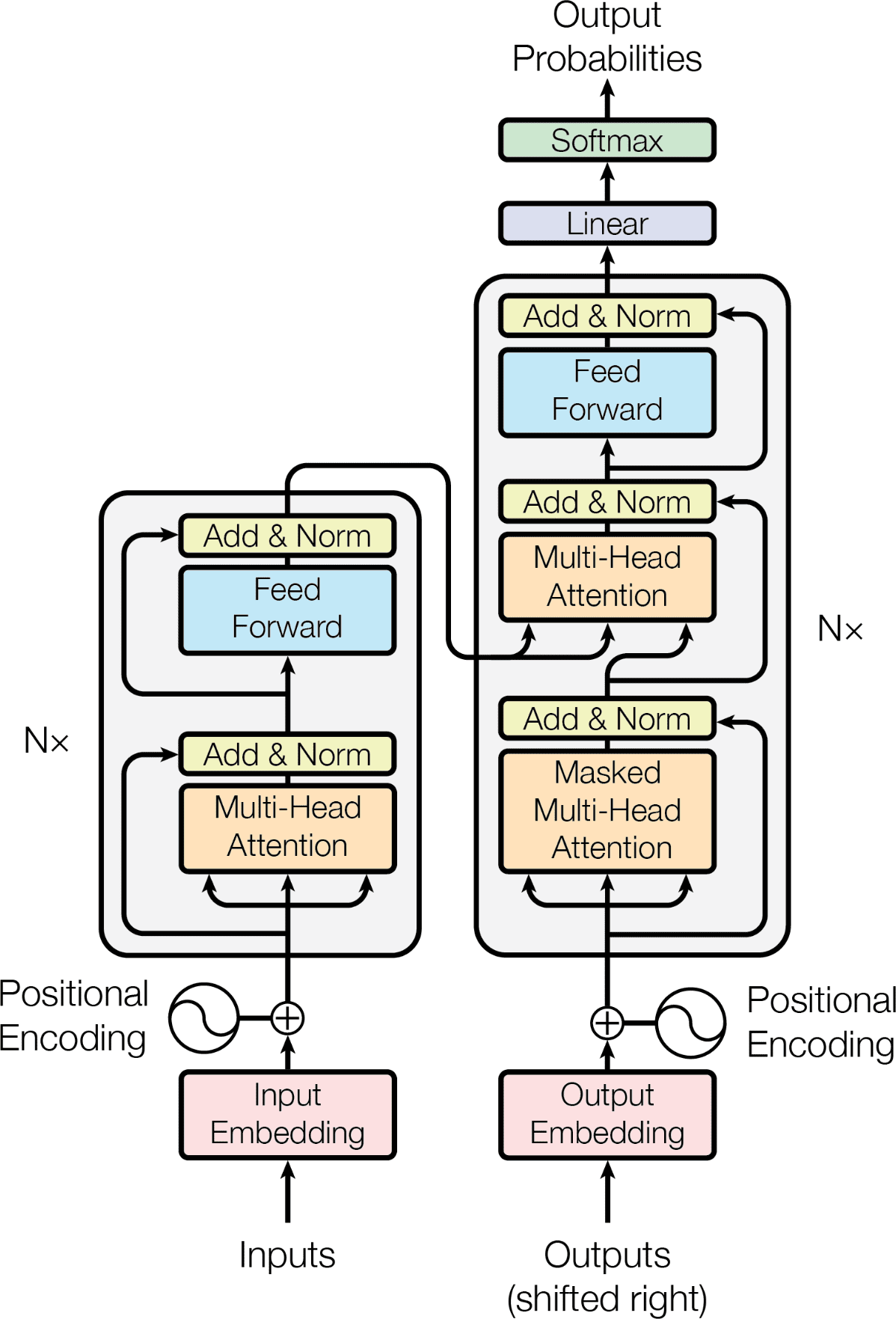


де: pos – позиція i – вимір

Синусоїдальний характер позиційних кодувань гарантує, що будь-які дві позиції матимуть унікальне кодування та що кодування мають бажані властивості, такі як лінійність. Використання функцій синусів і косинусів різних частот дозволяє моделі навчитися звертати увагу на відносні положення.

Архітектура трансформера складається з кодера та декодера. Завдання кодера в лівій половині полягає в тому, щоб зіставити вхідну послідовність з послідовністю неперервних представлень, які потім подаються в декодер. Декодер у правій половині отримує вихідні дані кодера разом із вихідними даними свого попередньго блоку для генерації вихідної послідовності.

Трансформера візуально виглядає так:



Кодер складається зі стека з N=6 ідентичних блоків. Кожен блок має 2 шари. Перший шар представлений механізмом багатоголової самоуваги, а другий – проста щільна нейронна мережа з ReLU активаційною функцією.

Другий шар — це щільна нейронна мережа прямого поширення, що складається з двох лінійних перетворень із активацією Rectified Linear Unit (ReLU) між ними:

N блоків кодера застосовують однакові лінійні перетворення до всіх слів у вхідній послідовності, але кожен блок використовує для цього різні параметри ваг (W1, W2) і ухилів (b1, b2).

Крім того, кожен із двох шарів має залишковий зв’язок (residual connection) навколо себе та нормалізацію. Тобто результат кожного шару дорівнює:

Result = LayerNorm(x + Layer(x)),

де Layer(x) — це функція, реалізована самим шаром.

Декодер також складається зі стека з N=6 ідентичних блоків. Перший шар отримує результат попереднього блоку декодера зі стаку як вхідні дані, доповнює їх позиційним кодуванням, та реалізує механізм багатоголової самоуваги. На відміну від кодера, який звертає увагу на всі слова вхідної послідовності, незалежно від їх позиції в послідовності, декодер звертає увагу лише на попередні слова. Отже, передбачення для слова в позиції може залежати лише від вихідних станів для слів, які стоять перед ним у послідовності. У механізмі багатоголової самоуваги це досягається шляхом введення маскування значень, що його отримують масштабованим множенням матриць Q і K. Це маскування реалізується шляхом придушення значень матриці, які в іншому випадку відповідали б забороненим зв’язкам:

Другий шар реалізує механізм багатоголової самоуваги, подібно до першого шара блоків кодера. На стороні декодера цей механізм отримує запити Q від попереднього шара декодера, а також ключі K та значення V з кодера. Це дозволяє декодеру звертати увагу на всі слова у вхідній послідовності.

Третій шар реалізує щільну нейронну мережу прямого поширення, подібну до тієї, що реалізована на другому шарі блоків кодера. Крім того, усі три шари на стороні декодера також мають залишкові зв’язки навколо них і нормалізацію.

ВИСНОВКИ ДО РОЗДІЛУ 1.

Ми почали розділ з встановлення актуальності задач обробки природної мови з фокусом на автоматичному розпізнавання мовлення. Була визначена ціль роботи: створення моделі для автоматичного розпізнавання українського мовлення та генерація транскрипції почутого. Потім був проведений аналіз предметної області. Відстежено виток галузі, зміну трендів з часом, причини змін.

Були досліджені основні етапи обробки природної мови: попередня обробка даних, моделювання. Були розглянуті такі техніки вилучення ознак як мел-частотні кепстральні коефіцієнти, перцептивне лінійне передбачення. Була встановлена роль нормалізації під час моделювання та були наведені основні її техніки: нормалізація Z-оцінки та довжини голосового тракту.

Далі були досліджені дві актуальні категорії моделей обробки автоматичного розпізнавання мовлення: традиційні методи машинного навчання (приховані моделі Маркова, моделі суміші Гауса) та методи глибинного навчання (рекурентні нейронні мережі, довга короткочасна пам’ять, моделі “послідовність-до-послідовності” та трансформер). Тут також були розглянуті загальні питання, що стосуються будь-якої галузі глибинного навчання: нейронні мережі, гіперпараметри, принципи розподілу набору даних, регуляризація, нарощення даних, техніки боротьби з перенавчанням.

Найбільша увага була приділена методам глибинного навчання, а саме трансформеру та його головної складової – механізму самоуваги. Саме на трансформер ми опиратимемося для створення нашої моделі.

РОЗДІЛ 2. СТВОРЕННЯ МОДЕЛІ

Перший крок – створення набору даних для тренування моделі. Автоматичне розпізавання усної мови полягає у перетворенні аудіо доріжки у транскрипцію (текст, субтитри). Тому набір даних буде складатися з аудіофайлів на вхід, та їх транскрипцій (субтитрів) у якості ярликів. Процес створення набору даних з нуля є дуже витратним процесом, адже навчання під наглядом потребує створення ярликів вручну, тобто людина повинна прослухати кожну аудіо доріжку та створити транскрипцію почутого. На щастя, у відкритому доступі вже існує багато наборів даних. У своїй більшості аудіо доріжки належать різним програмам з українського телебачення або є фрагментами аудіокниг. Для тренування нашої моделі ми скомпілюємо декілька таких наборів даних. Це допоможе моделі уникнути перенавчення (overfitting), адже дані належать різним джерелам і мовникам. Також буде використаний окремий набір даних для порівняння з існуючими моделями.

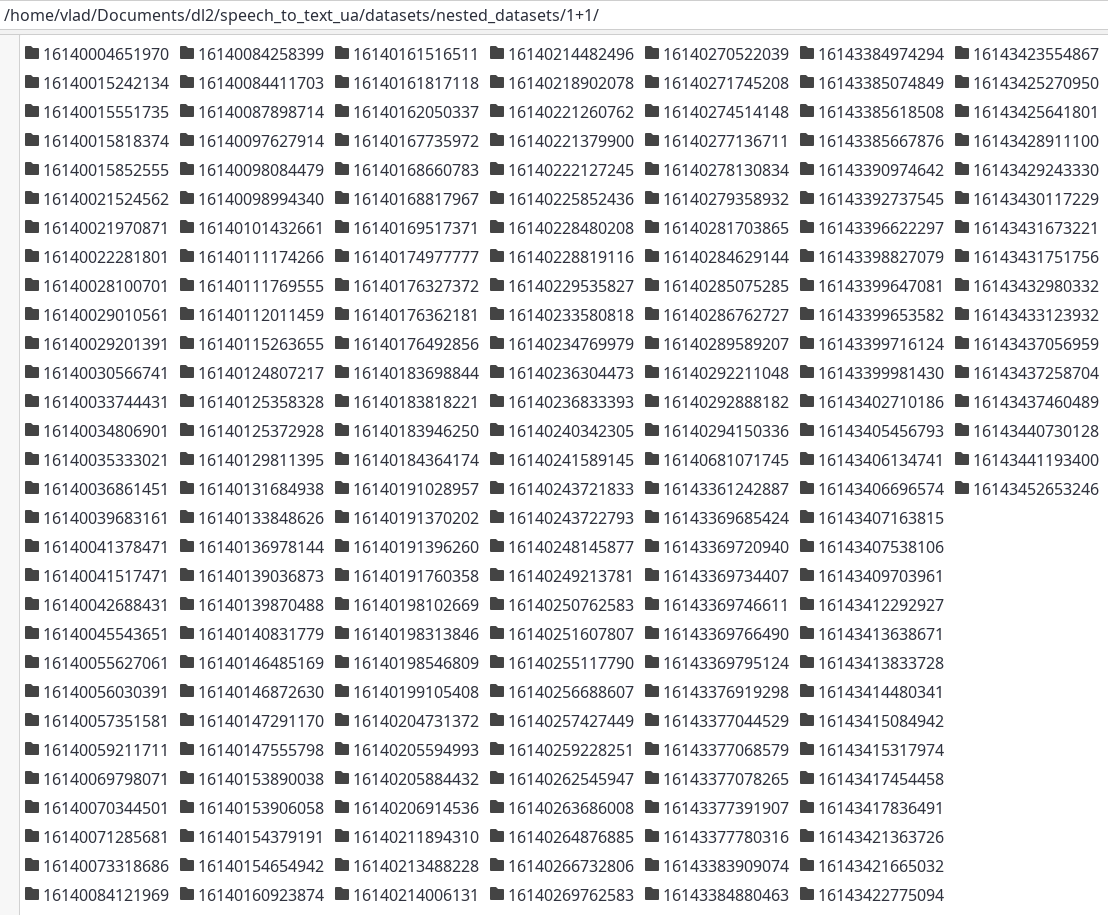
Початковий набір даних складатиметься з декількох піднаборів, кожний з яких матиме наступну структуру:

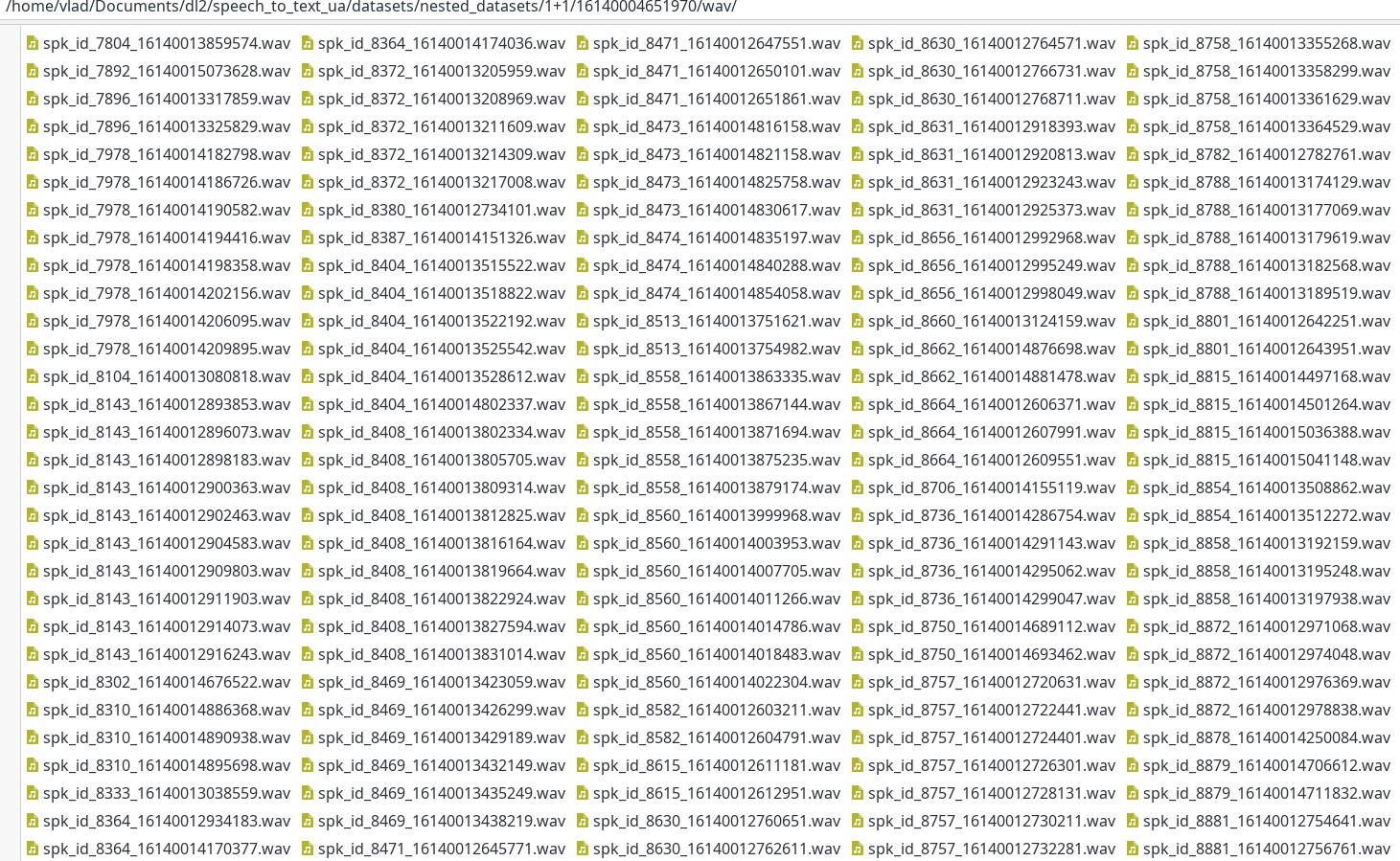
project\_root/datasets/nested\_datasets/{dataset\_name}/{episode}

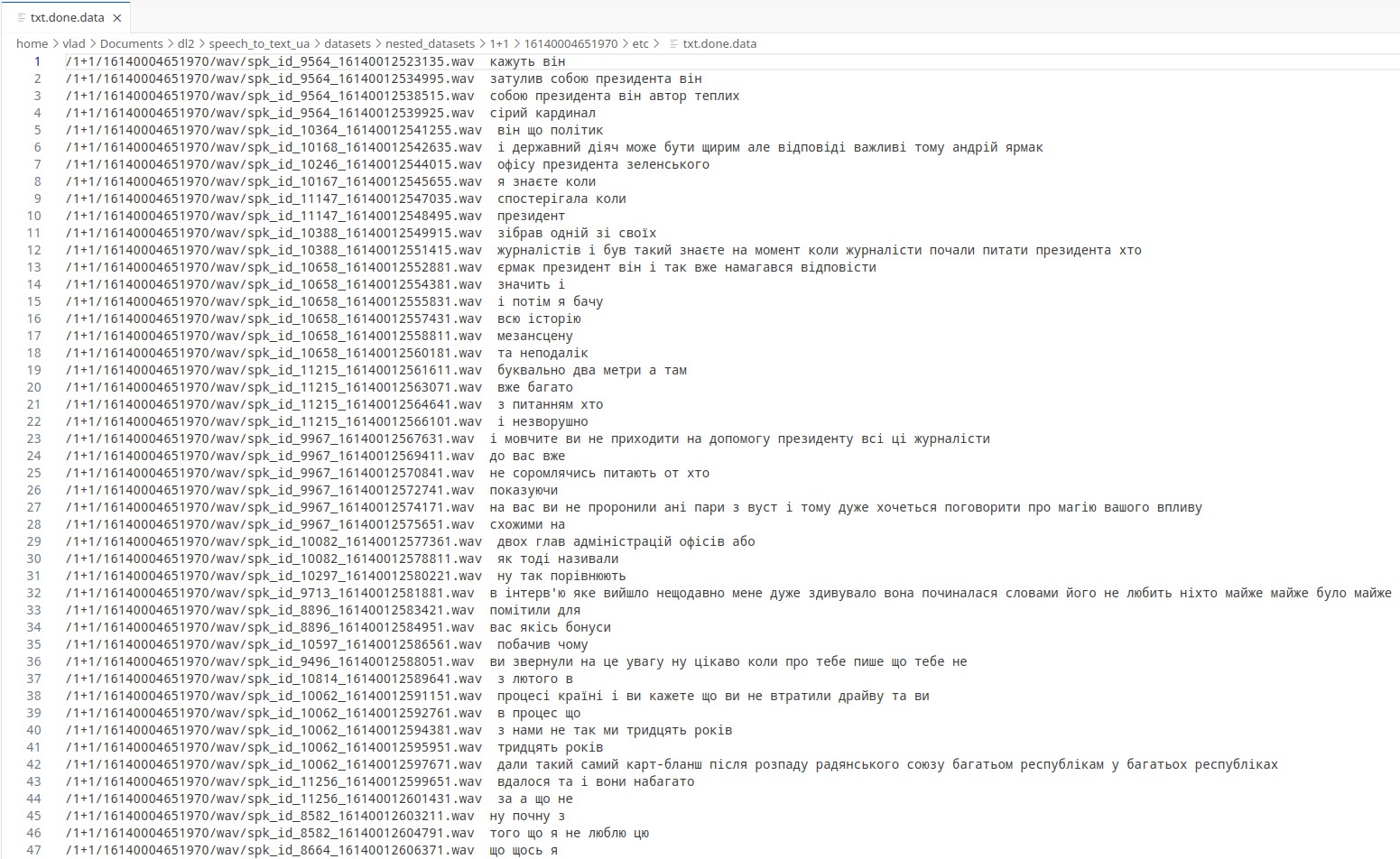
Всі дані знаходитимуться у папці datasets.

dataset\_name – назва конкретного набору даних, наприклад, 1+1, audiobooks, ukrainer тощо. Даних у кожному наборі достатньо багато, тому вони розбиті у папки на меньші піднабори – епізоди (episode). Назва епізоду, зазвичай, мітка часу unix у мілісекундах. Кожний епізод складається з папки аудіофайлів – wav, та папки транскрипцій – etc. Папка etc містить один файл – txt.done.data, що містить у собі транскрипції для усіх аудіофайлів з папки wav, по одному на лінію. Ми використовуватимемо цей файл для оцінки під час тренування моделі.

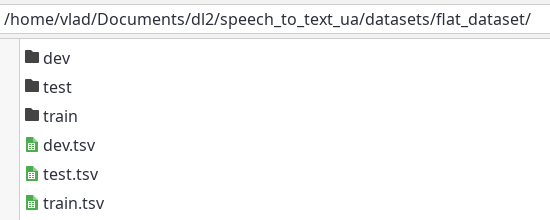
Можна побачити, що структура набору даних є доволі вкладеною, і підлаштовувати тренування під таку вкладеність буде незручно. Тож перед тренуванням ми перетворимо структуру на більш плоску. Саме тому початкові набори даних ми помістили у папку під назвою nested\_datasets.







Далі, за допомогою bash-скрипта, ми перенесемо дані у папку під назвою flat\_dataset. Структура цієї папки буде мати наступний вигляд:



Bash-cкрипт робить декілька операцій:

1. Рахує кількість аудіофайлів у wav папці
2. Розділяє аудіофайли на 3 частини: dev – 10%, test – 10%, train – 80%.
3. Переносить аудіофайли у відповідну папку: dev/test/train
4. Переносить ярлики (транскрипції) аудіофайлів з etc/txt.done.data файлу у відповідний dev/test/train.tsv файл, виправляючи шлях до файлу:

/1+1/1724664632/wav/spk\_id\_1\_2.wav → 1+1\_1724664632\_wav\_spk\_id\_1\_2.wav Файл скрипта має назву flatten\_dataset.sh і наведений у додатку.

Приклад використання flatten\_dataset:

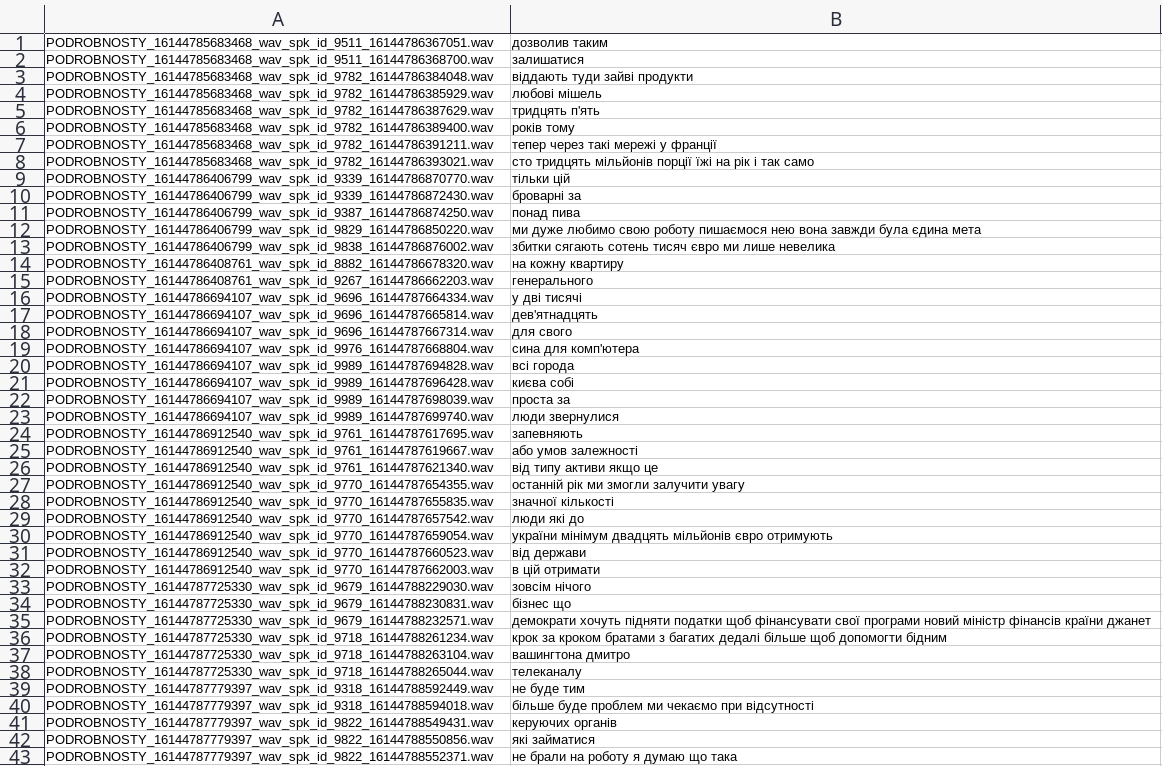
./flatten\_dataset {folder}

де folder - піднабір даних, наприклад 1+1.

В результаті аудіофайли з усіх під-наборів даних будуть злиті разом з ярликами в єдиний набір – flat\_dataset.

dev/test/train підпапка має наступний вигляд:

dev.tsv/test.tsv/train.tsv має наступний вигляд:



Деколи під час роботи flatten\_dataset.sh будуть виникати помилки, одна з найчастіших – помилка під час пошуку та перенесення ярлика для аудіофайлу: "Key '$new\_audio\_file' does not exist"

Існує декілька причин для цієї помилки.

Перша з них – шлях до аудіофайлу в etc/txt.done.data прописаний з помилками. Дуже поширеною помилкою є пропуск /wav частини шляху:

Помилка:

/AUDIOBOOKS/15\_річний\_капітан/spk\_id\_10000002\_1602308265177.wav Має бути:

/AUDIOBOOKS/15\_річний\_капітан/**wav**/spk\_id\_10000002\_1602308265177.wav

Коли таке трапляється, flatten\_dataset.sh не може співставити ярлик з аудіофайлом, що був переміщений до flat\_dataset.

Для виправлення цього створимо новий bash-скрипт: fix\_missing\_transcript.sh.

Він полягає у повторному проходженні кожної строки відповідного etc/txt.done.data файлу та пошуку аудіофайлів без /wav частинки в шлясі. Далі, для кожного збігу, ми знайдемо аудіофайл, що був перенесений до flat\_dataset, дізнаємось до якого ярликового файлу (dev.tsv/test.tsv/train.tsv) він повинен належати (за шляхом) і додамо його у відповідний ярликовий файл.

Для прикладу, що був наведений вище, fix\_missing\_transcript.sh буде використаний наступним чином:

./fix\_missing\_transcripts.sh AUDIOBOOKS 15\_річний\_капітан

Друга причина – відсутність ярликів для аудіофайлів з etc/txt.done.data. Ми не можемо використовувати такі дані під час тренування. Присутність даних без ярликів може погіршити результат тренування моделі, адже модель опирається на ярлики для розуміння, правильна відповідь чи ні. Такі дані краще перемістити у тимчасову папку для подальшого додавання ярликів або видалення.

Створимо новий bash-скрипт під назвою skip\_audios\_wo\_transcript.sh, що буде читати решту помилок з error.log файлу одну за одною, переконуватися, що у файла дійсно не має ярлика та переміщатиме його до папки unlabeled\_speech\_ds для подальшого опрацювання. Скрипт використовуватиметься наступним чином:

./skip\_audios\_wo\_transcript.sh

Перейдемо безпосередньо до архітектури моделі. За основу для нашої моделі ми візьмемо попередньо навчену модель та скористаємося технікою перенесення навчання [44].

Попередньо навчена модель — це збережена модель, що була попередньо навчена на великому наборі даних. Під час перенесення навчання попередньо навчена модель служить відправною точкою для навчання нової моделі для іншого, але пов’язаного завдання. Це дозволяє новій моделі використовувати знання та представлення, отримані з попередньо навченої моделі, що може пришвидшити процес навчання та покращити загальну продуктивність нової моделі.

Мотивацію перенесення навчання можна знайти в ідеї «Вчимося вчитися» (NIPS 95), де стверджується, що навчання з нуля (навчання tabula rasa) часто обмежене, і тому минулий досвід слід використовувати якомога більше. Наприклад, один раз ми дізналися, що тверде яблуко часто буває кислим, цей досвід може використовується, коли ми вибираємо груші: ми припускаємо, що тверді груші також кислі.

Модель, що була попередньо навчена на великому наборі даних для автоматичного розпізнавання усної англійської мови, можна переналаштувати для виконання розпізнавання усної української мови, шляхом навчання на меншому наборі даних (аудіофайлів), схожих на вихідний набір даних. Це потужна техніка, яка може значно зменшити кількість даних і час, необхідний для навчання моделі та підвищити її точність.

Взагалі, мови для задачі автоматичного розпізнавання мовлення розрізняють на дві категорії: мови з високим ресурсом та мови з низьким ресурсом. Ресурс тут означає кількість даних з ярликами яку можна знайти у вільному доступі. Прикладами мов з високим ресурсом є англійська, китайська, іспанська. Прикладами мов з низьким ресурсом є українська, білоруська, хорватська. Автоматичне розпізнавання усної мови має довгу історію досліджень; перші підходи використовували статистичні методи, такі як динамічне викривлення часу, а пізніше — приховані моделі Маркова. Приблизно в 2010-х роках підходи до глибоких нейронних мереж стали більш поширеними для моделей розпізнавання мовлення, що стало можливим завдяки доступності великих наборів даних («big data») і підвищеній обчислювальній продуктивності. Ранні підходи до глибокого навчання в розпізнаванні мовлення включали згорткові нейронні мережі, які були обмежені через їхню нездатність захоплювати послідовні дані, що пізніше призвело до розробки підходів Seq2seq, які включають рекурентні нейронні мережі, які використовували довгу короткочасну пам’ять.

Трансформери, представлені компанією Google у 2017 році (Attention Is All You Need), витіснили багато попередніх підходів до багатьох завдань машинного навчання та стали основною нейронною архітектурою в таких сферах, як мовне моделювання та комп’ютерне бачення; підходи слабкого керування тренуванням акустичних моделей були визнані на початку 2020-х років перспективними для розпізнавання мовлення з використанням глибоких нейронних мереж. Слабке керування полягає у комбінуванні керованого та некерованого навчання моделі. Цей підхід особливо актуальний коли даних дуже багато, але створювати ярлики затратно по часу або ресурсам.

Відповідно до звіту New York Times, у 2021 році компанія OpenAI вважала, що вичерпала джерела високоякісних даних для навчання своїх великих мовних моделей, і вирішила доповнити зібраний веб-текст транскрипціями відео та подкастів YouTube. Саме для цього завдання був розроблений Whisper (WSPSR – Web-scale Supervised Pre-training for Speech Recognition) [48, 49, 50].

Мотивація Whisper полягає у боротьбі з проблемою перетренування, яка властива моделям обробки мовлення [45]. Найбільш сучасні моделі розпізнавання мовлення використовували тільки парадигму керованого навчання і були натреновані на високоякісних але відносно невеликих наборах даних (~1000 годин). Такі моделі досягали точності, що перевершувала людську, але тільки на цих специфічних наборах даних, покладаючись на особливості даних, які не помітні людському сприйняттю. Як тільки модель зустрічала аудіо з іншого джерела, з іншими особливостями, точність падала. Вирішенням цієї проблеми є додавання великої кількості низькоякісних даних, тобто таких даних, що мають ярлики, які не були підтверджені експертом – людиною. Такий підхід називається слабке керування [51]. Він погіршує максимальну точність моделі, але значно покращує її стійкість до нових, раніше не бачених даних.

Whisper, є сильним кандидатом для роботи з мовами з низьким ресурсом, як-от українська, з низки причин наведених нижче:

* Whisper навчається на великому різноманітному наборі даних (680,000 годин) , що включає багато мов, у тому числі мови, що вважаються малоресурсними (частка неангломовних даних становить 1/5). Це дозволяє моделі досить добре працювати навіть на мовах з малою кількістю даних.
* Здатність Whisper працювати з декількома мовами означає, що він може виконувати розпізнавання навіть на нових для нього мовах (zero-shot learning). Це дозволяє моделі краще узагальнювати мови, які мають обмежені дані, використовуючи знання зі споріднених мов.
* Whisper розроблений бути стійким проти діалектів та акцентів, що допомагає при тренування на мовах з низьким ресурсом.

Виходячи з цього ми використаємо Whisper як базову модель для тренування автоматичного розпізнавання усної української мови. Розглянемо цю модель детальніше. Архітектура Whisper реалізована за допомогою трансформера, що складається з кодера та декодера. Вже перевірено, що ця архітектура добре підходить для масштабування [56].

Увесь звук повторно дискретизується (re-sampling) до 16 000 Гц, а 80-канальне представлення спектрограми Мела з логарифмічною величиною обчислюється на 25- мілісекундних вікнах із кроком у 10 мілісекунд. Для нормалізації особливостей вхідні дані масштабуються в діапазоні від -1 до 1 із приблизно нульовим середнім значенням у наборі даних перед навчанням. Кодер спочатку пропускає таке представлення даних через два згорткових шара з величиною фільтра 3 і GELU активаційної функції. Згортковий шар використовує фільтр, маленьку треновану матрицю, для проходження по матриці вхідних даних, та витягання конкретних ознак, таких як частотні компоненти, зміни в часі, ритм та темп, шуми та артефакти. Математично операція згортки полягає у розрахунку скалярного добутку між фільтром і вхідними значеннями. Результатом цієї операції є карта ознак, яка підкреслює, де саме виявлені конкретні ознаки у вхідних даних.

GELU – це більш плавна версія RELU.

До виходу згорткових шарів додаються синусоїдальні позиційні кодування, і після цього отриманий результат передається до кодера трансформера. Трансформер використовує до-активаційні залишкові блоки (pre-activation residual blocks), і на виході останнього шара кодера застосовується нормалізація. Залишкові блоки полягають у додаванні результатів прямого шляху та пропускного шляху для того, щоб зарадити проблемі зникнення/вибуху градієнта. Прямий шлях – це звичайне проходження декількох шарів нейронної мережі. Пропускний шлях (skip connection) пропускає декілька шарів нейронної мережі. У до-активаційних залишкових блоках активаційна та нормалізаційна функції застосовуються до проходження прямого шляху, а не після [53].

Декодер використовує попередньо вивчені позиційні кодування та пов’язане представлення токенів вводу-виводу (Press & Wolf, 2017) [54], що полягає у використанні єдиної матриці вкладень (embeddings) для перетворення вхідних та вихідних токенів (слів/фонем) на вектори вкладень. Вектори вкладень зображують токени у багатовимірному просторі, де семантично чи контекстуально схожі лексеми поміщаються поряд. Цей підхід зменшує кількість параметрів які потрібно тренувати, і тим самим покращує точність моделі. Кодер та декодер мають однакову ширину (кількість нейронів в усіх шарах) та кількість блоків трансформера (механізму самоуваги).

Whisper тренований на аудіофайлах довжиною до 30 секунд і не може приймати більш довгі дані. Це не є проблемою для академічних наборів даних, тому що більшість і них складається з коротки фраз, але цей факт створює проблему для застосування у реальному світі на практиці, адже зачасту аудіофайли виміряються хвилинами або навіть годинами. Для боротьби з цим Whisper розбиває довгі аудіофайли на відрізки та розглядає не більше 30 секунд за раз. Ключову роль тут грає променевий пошук [55].

Променевий пошук – це евристичний алгоритм пошуку, що досліджує граф, розширюючи найперспективніші вузли в обмеженому їх наборі. У контексті автоматичного розпізнавання мовлення цей алгоритм вибирає яке наступне слово слід сгенерувати. Його перевага над звичайним жадібним алгоритмом полягає у тому, що променевий пошук розглядає N (величина проміня) кращих варіантів замість знаходження одного найкращого варіанта. Для довших транскрипцій прийняття рішень на основі локальних ймовірностей (як у жадібному пошуку) може призвести до недалекоглядних виборів, що призведе до безглуздих або неточних транскрипцій далі в послідовності.

Whisper має 5 конфігурацій, що відрізняються за розміром: кількістю тренувальних параметрів, кількістю шарів, шириною шарів, кількістю голов механізму самоуваги. Деякі з них також мають по два варіанти – англомовний та багатомовний, а деякі тільки один – багатомовний. У таблиці наведено інформацію про кожну з основних конфігурацій:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Назва | Шари | Ширина | Голови | Параметри | Англ. вар. | Багатомовний вар. |
| Tiny | 4 | 384 | 6 | 39 млн. | так | так |
| Base | 6 | 512 | 8 | 74 млн. | так | так |
| Small | 12 | 768 | 12 | 244 млн. | так | так |
| Medium | 24 | 1024 | 16 | 769 млн. | так | так |
| Large | 32 | 1280 | 20 | 1.55 млрд. | ні | так |

Вибір конфігурації полягає у співставленні бажаної точності проти часу тренування та наявних ресурсів. Чим більше параметрів має модель тим більше часу та обчислювальних ресурсів вона потребує для тренування, але і точність моделі буде більшою.

Tiny і Base зазвичай використовують у випадках, коли швидкість більш важлива за точність. Ці конфігурації використовуються лише для простих задач, і погано справляються з більш складними задачами, такими як наявність шуму або акценту у вимовника.

Конфігурація small вважається золотою серединою, адже вона має відносно малий розмір, але, назважаючи на це, справляється зі складними задачами, описаними раніше.

Конфігурації medium та large використовують коли швидкістю можна знехтувати на користь додаткової точності. Тренування моделей на основі таких конфігурацій потребує спецільного обладнання, у користувацькому середовищі тренувати такі моделі практично неможливо.

Для нашої моделі ми використаємо конфігурацію small, тому що вона дає достатню точність і не потребує надвеликих ресурсів та часу для тренування.

Перейдемо безпосередньо до процесу тренування моделі [57].

Спочатку потрібно визначити залежності проєкту. Мова python є стандартом для будь-якої задачі машинного навчання, автоматичне розпізнавання мовлення не є виключенням. Ми використаємо декілька популярних бібліотек для пришвидшення роботи: datasets[audio] від платформи huggingface для завантаження та підготовки набору даних, transformers та accelerate від цієї ж платформи для завантаження та тренування попередньо навченої моделі Whisper. Huggingface підтримує обидві найбільш популярні бібліотеки машинного мавчання: Tensorflow та Pytorch. Ми використаємо Pytorch. Для обробки аудіофайлів нам знадобиться бібліотека soundfile, а оцінювати точністі моделі під час тренування нам допоможе бібліотека jiwer.

Усі залежності можна виразити через наступні команди:

!pip install --upgrade pip

!pip install --upgrade datasets[audio] transformers accelerate evaluate jiwer

Почнемо створювати скрипт під назвою fine\_tune.py для тренування моделі. Спочатку завантажимо набір даних, що ми зібрали локально [47]. Для цього використаємо метод бібліотеки datasets під назвою load\_dataset. Для того аби завантажити локальний набір даних недостатньо просто вказати шлях до набору даних, натомість потрібно вказати шлях до файлу, що містить спеціальних клас, який відповідатиме за розподіл повного набору даних на тренувальні частини: train/val/test.

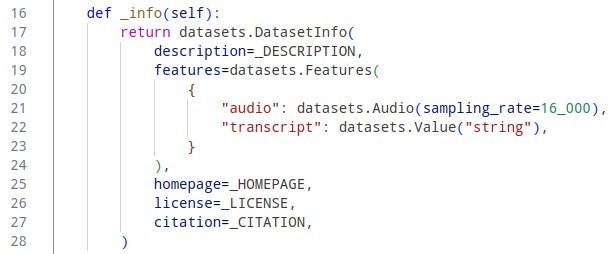
ds = load\_dataset("./flat\_dataset.py", trust\_remote\_code=True)

flat\_dataset.py файл має містити клас, що наслідує datasets.GeneratorBasedBuilder:

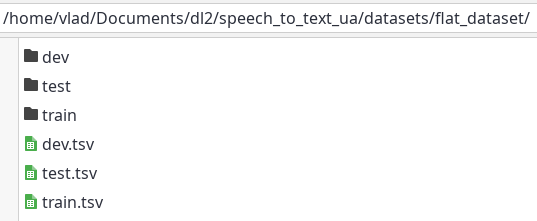
class FlatDataset(datasets.GeneratorBasedBuilder):

Нам потрібно перевизначити 3 методи цього базового класу:

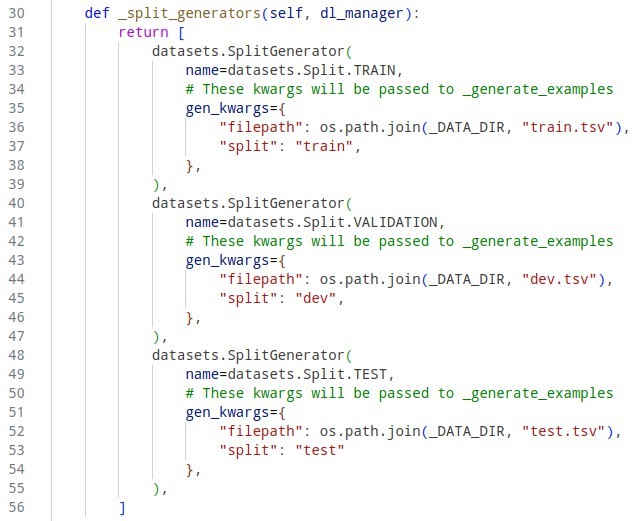
1. \_info містить інформацію про набір даних (опис, ліцензія, особливості кожного зразка).
2. \_split\_generators завантажує набір даних в оперативну пам’ять та визначає частини train/val/test.
3. \_generate\_examples генерує зразок для кожної частини. Кожний зразок повинен містити особливості вказані в \_info методі.



Важливим є код features на строках 19-24. Тут ми визначаємо особливості для кожного тренувального екземпляра, а саме аудіофайл із частотою дискретизації (sample rate) 16kHz та транскрипцію почутого. Про важливість частоти дискретизації буде сказано трохи пізніше.

Нагадаємо, що структура папки набору даних виглядає наступним чином:

Ця структура допоможе нам визначити метод \_split\_generators:

де \_DATA\_DIR = "datasets/flat\_dataset"

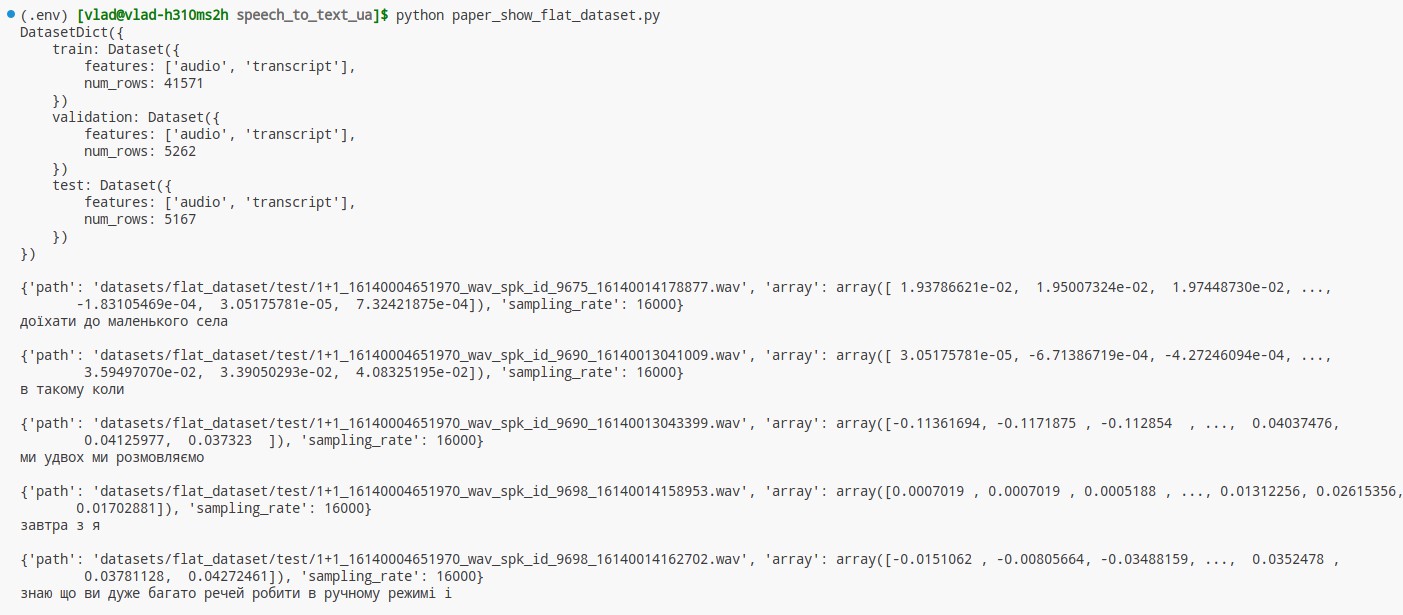
\_split\_dataset знаходить три тренувальні частини за допомогою файлів транскрипцій:

* + train частина – datasets/flat\_dataset/train.tsv
  + dev(eval) частина – datasets/flat\_dataset/dev.tsv
  + test частина – datasets/flat\_dataset/test.tsv
  + Нарешті, останній метод:



\_generate\_examples відкриває кожний файл транскрипцій, що був визначений як filepath в методі \_split\_generators, проходить по кожному рядку, і розбиває його на 2 частини: шлях до аудіофайлу і транскрипцію, відповідно до особливостей (features) визначених у \_info методі.

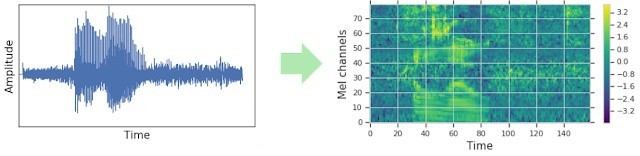
Перевіримо результат завантаження набору даних та покажемо перші 5 екземплярів із test частини:



Процес виконання завдання автоматичного розпізнавання мовлення можна розділити на 3 кроки:

1. Виявлення особливостей та пре-обробка аудіо даних за допомогою екстрактора.
2. Відображення послідовності звуку в послідовність тексту (транскрипція почутого) – моделювання.
3. Відображення вихідної послідовності токенів у текстове представлення за допомогою токенізатора.

Мовлення можна представити за допомогою одновимірного масиву, що змінюється з часом. Значення масиву у кожному кроці часу – це амплітуда сигналу у цій точці часу. З однієї інформації про амплітуду можна отримати спектр частот (за допомогою перетворення Фур’є) і відтворити усі акустичні особливості. Так як мовлення безперервне, воно містить безкінечну кількість значень амплітуд. Це є проблемою для комп’ютерів, тому що вони очікують дискретні масиви даних. Тому, ми дискретизуємо мовний сигнал, відбираючи значення з нашого сигналу з фіксованими кроками часу. Цей процес називається семплінгом. Інтервал з яким відбираються значення називається частотою дискретизації і вимірюється у герцах. Дискретизація з вищою частотою забезпечує кращу апроксимацію безперервного мовного сигналу, але також вимагає зберігання більшої кількості значень за секунду. Дуже важливо, щоб частота дискретизації наших аудіофайлів відповідала частоті дискретизації, яка очікується нашою моделлю, оскільки аудіосигнали з різними частотами дискретизації мають дуже різні розподіли. Ми повинні використовувати тільки екземпляри аудіо із корректною частотою дискретизації, нехтування цим правилом призведе до неочікуваних результатів. Наприклад, якщо взяти аудіофайл із частотою дискретизації 16kHz і прослухати його із частотою 8kHz, то звукова доріжка звучатиме, ніби вона у половинній швидкості. Whisper очікує аудіофайли з частотою дискретизації 16kHz, тому потрібно впевнитися, що дані відповідають очікуванню. Саме тому у методі \_info ми визначаємо sampling\_rate=16\_000.

Екстрактор Whisper виконує дві операції. По-перше він додає до кінця екземпляра нулі, що називається оббивкою (padding) або відрізає зайві секунди з кінця, таким чином, щоб кожний екземпляр мав довжину рівну 30 секундам. В інших моделях на цьому кроці також потрібно маскувати оббивку, щоб модель могла відрізнити справжні паузи у мовленні та штучні, що були додані оббивкою. Для Whisper цього робити не потрібно, через те що екземпляри стандартизовані до фіксованої довжини (30 секунд), чого не роблять інші моделі. Друга операція екстрактора Whisper – перетворення стандартизованого аудіо на логарифмічні спектрограми Мела.

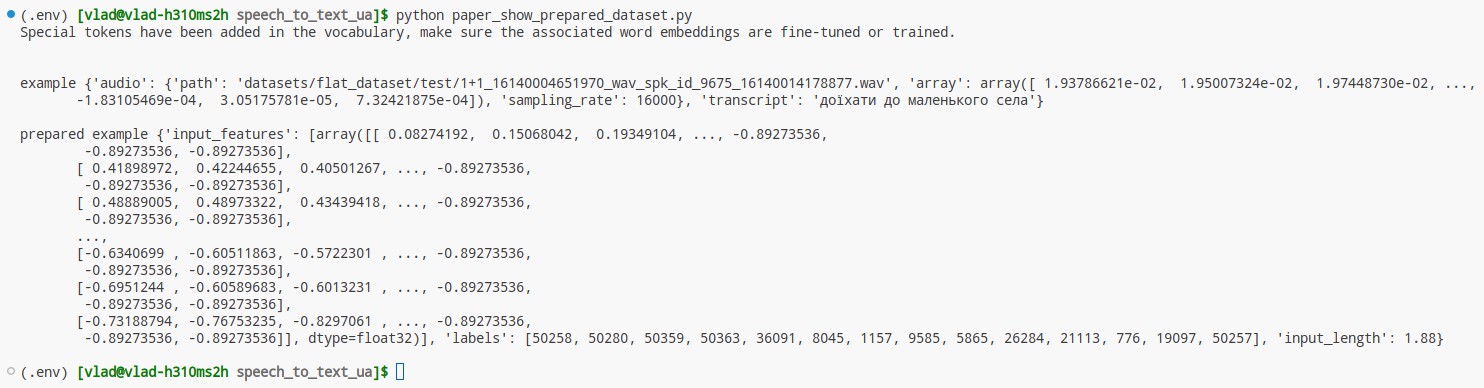
Приклад спектрограми [58]:

Уздовж осі y розташовані канали Мела, які відповідають певним діапазонам частот. По осі х - час. Колір кожного пікселя відповідає логарифмічній інтенсивності цього діапазону частот у заданий час. Логарифмічна спектрограма Мела є формою вхідних даних, очікуваних моделлю Whisper. Платформа Huggingface включає Whisper у свою бібліотеку transformers, а також пов’язаних екстрактор і токенізатор. Для зручності ці два об’єкти поєднані в один клас – WhisperProcessor. Ми використовуватимемо цей клас як для при-обробки даних (input\_features) так і для трансформації транскрипції у токени (labels).

Повернемося до fine\_tune.py і створимо функцію для пре-обробки завантажених даних для подачі у Whisper.

“openai/whisper-small” – це назва базової моделі з вибраною нами конфігурацією small.

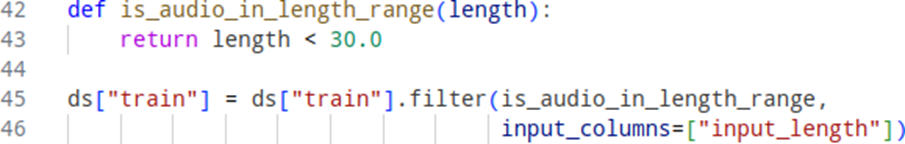
Результат роботи (input\_features, labels) функції prepare\_dataset на одному екземплярі:



На усьому наборі даних використання prepare\_dataset виглядатиме так



Нарешті, ми відкинемо усі аудіозразки тривалість яких перевищує 30 секунд. Інакше ці зразки будуть скорочені екстрактором Whisper, що може вплинути на стабільність навчання. Саме для цього ми визначили додаткове поле на екземплярі у функції prepare\_dataset – input\_length. Ми визначаємо функцію, яка повертає True для зразків, з тривалістю менше 30 секунд, і False для тих, що довші

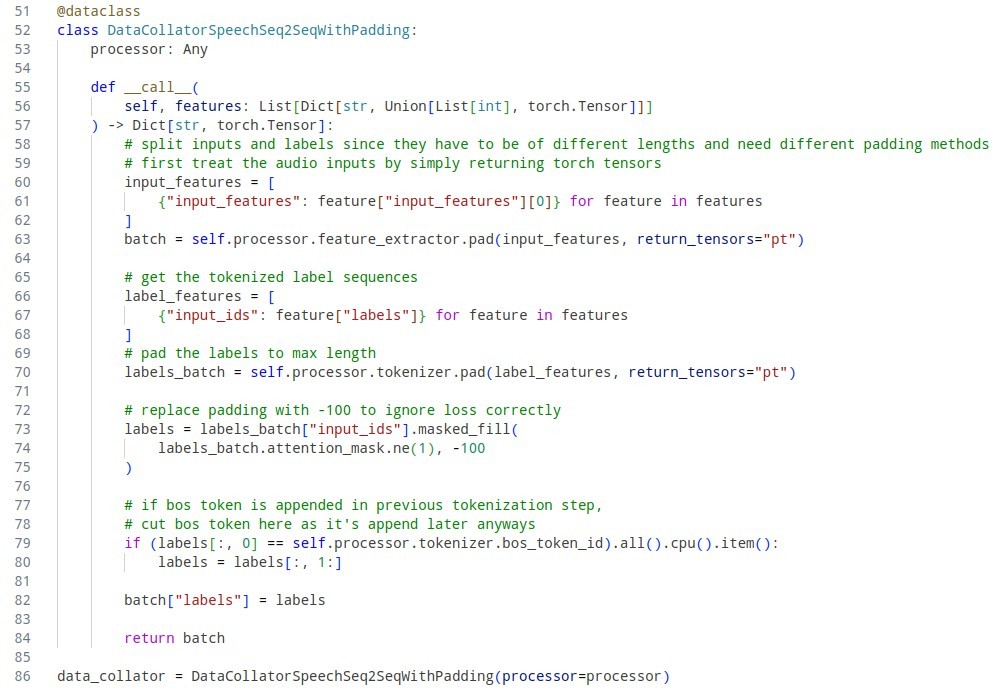


Тренувальні дані готові. Далі перейдемо безпосередньо до процесу тренування моделі. Для того щоб почати тренувати модель потрібно зробити наступне:

1. Створити клас – колатор даних (data collator, збірник даних). Колатор даних бере попередньо оброблені дані та перетворює їх на тензори PyTorch для моделі.
2. Визначити метрику оцінки: під час тренування ми використовуватимемо метрику під назвою частота помилки слова (Word rate error – WER), для цього визначимо функцію compute\_metrics.
3. Завантажити попередньо навчену модель на певній контрольній точці (checkpoint). Спочатку це буде базовий Whisper, згодом контрольна точка нашої моделі з попередньої сесії тренування.
4. Визначити налаштування тренування за допомогою гіперпараметрів.
5. Перевірити результат тренування на test – відкладеній частині тренувального набору даних.

Колатор (збірник) даних обробляє input\_features за допомогою екстрактора, а labels – токенізатора. Так як input\_features вже мають оббивку до 30 секунд і представлені у вигляді логарифмічних спектрограм Мела, усе що потрібно зробити – перетворити їх у зручний для тренування формат - тензор бібліотеки Pytorch.

З іншого боку поле labels не має оббивки. Тому спершу потрібно її додати, а потім замаскувати (змінити значення оббивки з 0 на -100), для того щоб оббивка не впливала на результат вирахування функції втрат. На відміну від input\_features тут маскування потрібно, адже ці значення (токени) відповідають словам транскрипції і не мають фіксованої довжини, як от вхідні дані, що завжди дорівнюють 30 секундам.

Для labels на цьому кроці також варто видалити символ початку речення – bos (beginning of sentence), тому що він буде доданий знову під час тренування. Для колатора даних знову використаємо класс WhisperProcessor. Код виглядатиме так:

Далі визначимо метрику оцінки яку ми будемо використовувати для валідаційного набору даних. Частота помилки слова – це стандарт для задач автоматичного розпізнавання мовлення. WER вимірює, наскільки точно модель робить транскрипцію порівнюючи її з коректною транскрипцією. Формула для частоти помилки слова має наступний вигляд:

де S – кількість замін (слів, що були неправильно вгадані моделлю),

D – кількість видалень (слів, яких модель не змогла додати в транскрипцію), I – кількість вставок (слів, яких модель додала в транскрипцію без потреби), N – загальна кількість слів у коректній транскрипції.

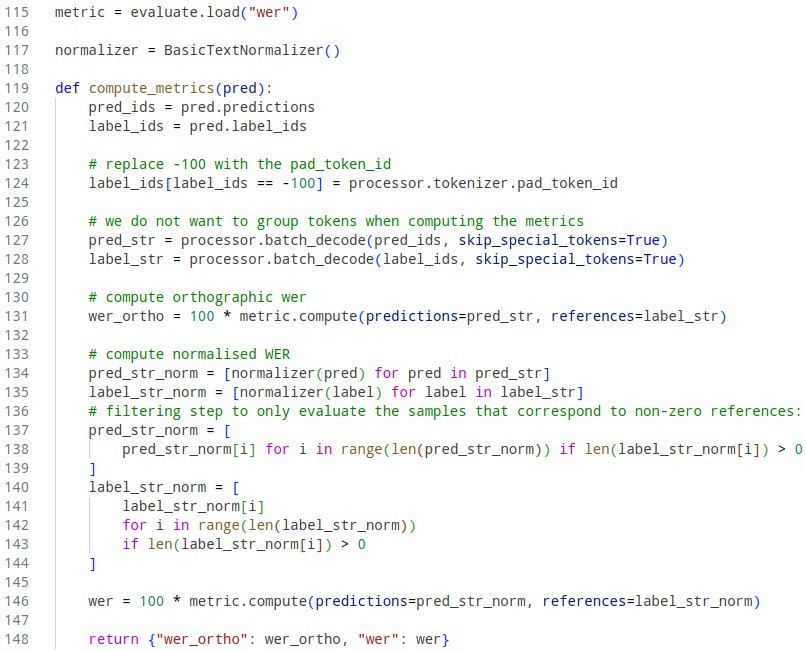
Чим меньший показник WER тим вища точність моделі. WER також дозволяє проаналізувати який саме тип помилки є слабкою стороною моделі: заміни, видалення чи вставки.

Функція compute\_metrics буде приймати на вхід прогноз з моделі і видавати метрику WER для цього результату. Прогноз складається з двох полів:predictions – транскрипція, що була сгенерована моделлю (у вигляді id токенів), label\_ids – коректна транскрипція для порівняння (у вигляді id токенів).

В першу чергу потрібно замінити усі замасковані значення (-100) в label\_ids на спеціальну константу pad\_token\_id, бо -100 було потрібно для коректного розрахунку функції втрат (допомога в ігноруванні оббивки), але на кроці розрахунку WER це значення є проблематичним, адже модель не вміє його порівнювати з корректною транскрипцією і через це WER буде звітувати про гірший результат ніж є насправді.

Наступним кроком є конвертація прогнозу та порівняльної транскрипції у строки. Далі ці строки використовуються для вирахування WER. Альтернативно тут можна вирахувати нормалізований WER, тобто видалити зі строк усю пунктуацію та звести букви в один реєстр і вирахувати значення WER на них. Нормалізованому WER насправді віддається перевага, адже він допомагає зробити акцент на покращенні мовлення, ігноруючи орфограчні відмінності

.Повний код compute\_metrics:

Для видалення орфографії використовується BasicTextNormalizer. Наступний крок – це завантаження моделі



На даному етапі \_PREV\_MODEL\_NAME дорівнюватиме базовому збереженню моделі, а саме “openai/whisper-small”. Далі, після кожного тренувального чиклу ми будемо робити свої збереження і використовувати вже їх для поновлення тренування.

Далі слід визначити гіперпараметри для тренування. Початкова конфігурація вигляла наступним чином



де

output\_dir – папка для збереження параметрів моделі.

per\_device\_train\_batch\_size – розмір партії (кількість екземплярів) для тренування на одному ядрі.

gradient\_accumulation\_steps – визначає через скільки кроків тренування оновлювати ваги. Якщо значення більше 1, то оновлення проводиться через визначену кількість кроків за допомогою акумульованого значення градієнту. Корисно у ситуаціях коли потрібно збільшити розмір партії без збільшення використання пам’яті.

learning\_rate – швидкість навчання (зміни параметрів). Використовується оптимізатор під назвою AdamW (Adaptive moment estimation Weight decay). AdamW є покращенням Adam у тому, що він відокремлює спад ваги від оновлення градієнта, застосовуючи його безпосередньо до ваг, що призводить до кращої регуляризації та продуктивності.

lr\_scheduler\_type – коректує зміну швидкості навчання в часі. Значення “constant\_with\_warmup” означає що зміни швидкості буде збільшуватися n-кроків до значення вказаного в гіперпараметі learning\_rate, де n – кількість кроків визначена у гіперпараметрів під назвою warmup\_steps.

max\_steps – загальна кількість кроків тренування партій. Цей гіперпараметр, на відміну від epoch, вказує, що тренування може зупинитися без повного проходження набору даних: max\_steps \* per\_device\_train\_batch\_size екземплярів буде протреновано.

gradient\_checkpointing – контрольні точки градієнта переобчислюють проміжні значення активацій (виходу з шара) під час зворотнього поширення, які зазвичай зберігаються під час прямого проходження. Це допомагає зменшити об’єм пам’яті, що потрібний моделі для тренування, у компроміс швидкості тренування.

fp16 – це 16-бітний формат із плаваючою комою, який економить пам’ять і підвищує ефективність обчислень.

fp16\_full\_eval – пришвидшує та оптимізує оцінку за допомогою 16-бітного формату, у компроміс точності.

eval\_strategy – steps або epoch.

per\_device\_eval\_batch\_size - розмір партії (кількість екземплярів) для валідації на одному ядрі.

predict\_with\_generate – дозволяє моделі генерувати вихідну транскрипцію під час валідації чи прогнозу. Коли значення Truе, модель використовуватиме такі стратегії як променевий пошук для генерації тектового результату, замість виведення сирих даних - ймовірностей токенів усіх слів/фонем у словнику без переведення їх у текст.

generation\_max\_length – визначає максимальну довжину вихідної транскрипції.

save\_steps – кількість кроків (партій) перед збереженням параметрів моделі на диск.

eval\_steps – кількість кроків (партій) перед оцінкою точності моделі на валідаційній частині даних.

logging\_strategy – логування інформації стосовної до тренування моделі. Значення: steps | epoch | no .report\_to – платформи для писання логів.

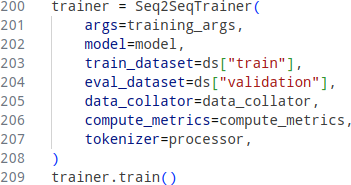
load\_best\_model\_at\_end – True щоб після завершення тренування зберігати параметри моделі, що дали найкращий результат. Інакше зберігається параметри останнього циклу.

metric\_for\_best\_model – метрика для порівняння моделей для load\_best\_model\_at\_end. За замовчуванням використовує функцію втрат.

greater\_is\_better – True якщо більше значення з metric\_for\_best\_model означає вищу точність моделі для load\_best\_model\_at\_end.

push\_to\_hub – True щоб завантажувати збережену модель на платформу Hugging face.

Нарешті, почнемо тренування моделі. Для цього використаємо класс Seq2SeqTrainer платформи Hugging face



Починаємо з тестового пробігу. Для нього візьмемо невелику кількість екземплярів (41,571 + 5,262 + 5167 = 52,000) відносно усього набору даних (~ 1,926,000 екземплярів). Для тренування моделі використаємо комп’ютер з наступної конфігурацією:

* Процесор: Intel Core i5-8400, 2.8 GHz x 6.
* Відеокарта: NVIDIA GeForce GTX 1660 Ti 6GB.
* Оперативна пам’ять: 32 GB.

Результат завантаження даних:

41571/41571 [49:48<00:00, 13.91 examples/s]

5262/5262 [06:15<00:00, 14.01 examples/s]

5167/5167 [06:10<00:00, 13.94 examples/s]

Можемо бачити, що на даному комп’ютері скрипту знадобилося 49 хвилин і 48 секунд щоб завантажити тренувальну частину (41,571) того невеликого набору даних. 6 хвилин і 15 секунд щоб завантажити валідаційну (5,262) та ще 6 хвилин і 10 секунд щоб завантажити тестову (5,167) порції. Процес завантаження зайняв так багато часу через prepare\_dataset функцію. Але результат цієї операції кешуються, тому повторний запуск скрипту вже омине ці обчислення.

Із цього заміру випливає, що на обробку одного екземпляру йде приблизно 0.0718 секунд. Тобто тільки для обробки усього набору нам знадобиться 1,926,000 \* 0.0718 ~ 38.5 годин безперервної роботи скрипту. Також потрібно врахувати, що комп’ютеру може забракнути оперативної пам’яті, щоб просто завантажити увесь набор даних, не кажучи вже про його тренування.

Результат тренування даних:

4% | 140/4000 [1:34:49<42:50:45, 39.96s/it]

За півтори години вдалося протренувати лише 4% від усього об’єму. Потім процес було перервано. З цього випливає, що на тренування 100% об’єму невеликого набору даних піде 44 години і 24 хвилини, а на тренування усього набору даних знадобиться набагато більше часу.

Із цих спостережень стає ясно, що наш комп’ютер недостатньо потужний для тренування моделі розпізнавання мовлення. Потрібно розглянути альтернативи, а також техніки оптимізації тренування.

Альтернативою для локальних обчислень є хмарні платформи для машинного навчання. Однією з провідних платформ є Google Colab. Такі платформи дають доступ до більш потужних відеокарт, що призначені саме для задач тренування моделей глибинного навчання. До таких відеокарт належать NVIDIA V100 та A100. Окрім загального покращення продуктивності (більше пам’яті, вища швидкість передачі даних, більше CUDA ядер тощо), такі відеокарти оснащені тензорними ядрами (Tensor Core) для пришвидшення матричних операцій, необхідних для паралелізації тренування моделей. Відеокарти загального призначення, такі як GeForce GTX 1660 Ti, не мають тензорних ядер взагалі, і тому сильно поступаються V100/A100.

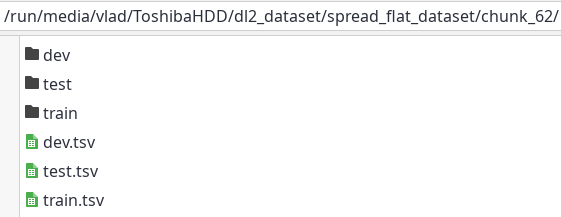
Для того, щоб користуватися набором даних у хмарному середовищі, їх потрібно зробити доступними, завантаживши на сервер. Для цього використаємо хмарний сервіс для зберігання даних Google Drive. Цей варіант добре інтегрований з Google Colab, адже обидва сервіси розроблені однією компанією. Для платформи Google Colab наш скрипт для тренування fine\_tune.py потрібно переробити у формат Colab Notebooks (.ipybn). Цей формат зручний тим, що .py файл розбивається на блоки і кожний блок коду можна запускати окремо. Наш fine\_tune.py зазнає мінімальних змін при перенесенні. Все що потрібно додати це невеликий блок коду для під’єднання Google Drive папки з набором даних до сессії Google Colab

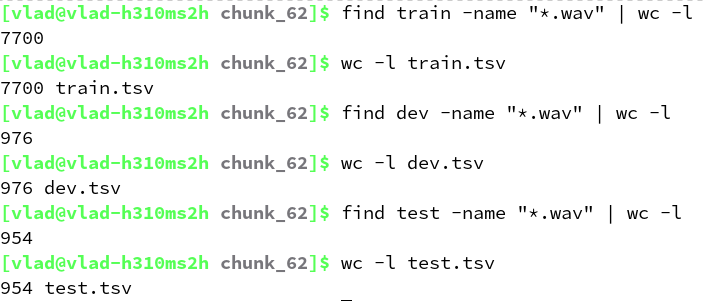
Наш набір даних занадто великий для прямого завантаження на такі сервіси як Google Colab, адже він важить понад 150 гігабайтів. Просто зменшувати набір даних небажано, адже чим більше різноманітних даних, тим краще модель буде узагальнювати на нових, раніше не бачених, екземплярах. Краще протренувати більше даних на меньшу кількість циклів, ніж навпаки, адже тоді модель може пристосуватися до тренувального набору даних (overfitting). Але для тренування має сенс розбити набір даних на менші піднабори-шматки: по ~7,7 тисяч тренувальних екземплярів, ~1 тисячу валідаційних та ~1 тисячу тестових екземплярів. Вийде приблизно 200 партій, які буде набагато зручніше завантажувати та тренувати, особливо на Google Colab.

Створимо скрипт під назвою spread\_out\_flat\_dataset.sh для розбиття набору даних на шматки. Цей скрипт прийматиме один аргумент – split = test | train | dev, що відповідатиме розподілу даних (тренувального / валідаційного / тестувального). Тож загалом скрипт потрібно буде викликати 3 рази. Скрипт працює наступним чином:

1. Визначається кількість .wav файлів у піднаборі
2. Розраховується кількість файлів піднабору, що потрібно перемістити у кожний шматок з розрахунком, що буде 200 шматків.
3. Перенести розраховану кількість файлів у кожну папку шматка. Кожний шматок, як і повний набір даних, складатиметься з папки аудіофайлів і файла транскрипцій.

Усі шматки будуть поміщені в нову папку під назвою spread\_flat\_dataset.

Результат роботи скрипту



Щоб пришвидшити виконання скрипту, його можна запустити у паралелі. Для цього створимо 3 копії скрипту з наступними назвами:

spread\_out\_flat\_dataset\_2.sh, spread\_out\_flat\_dataset\_3.sh, spread\_out\_flat\_dataset\_4.sh.

Далі змінимо частину скрипту, що відповідає за проходження по шматкам

for ((i = 0; i < 200; i++)); do на

for ((i = 0; i < 50; i++)); do для оригіналу spread\_out\_flat\_dataset.sh

for ((i = 50; i < 100; i++)); do для копії spread\_out\_flat\_dataset\_2.sh

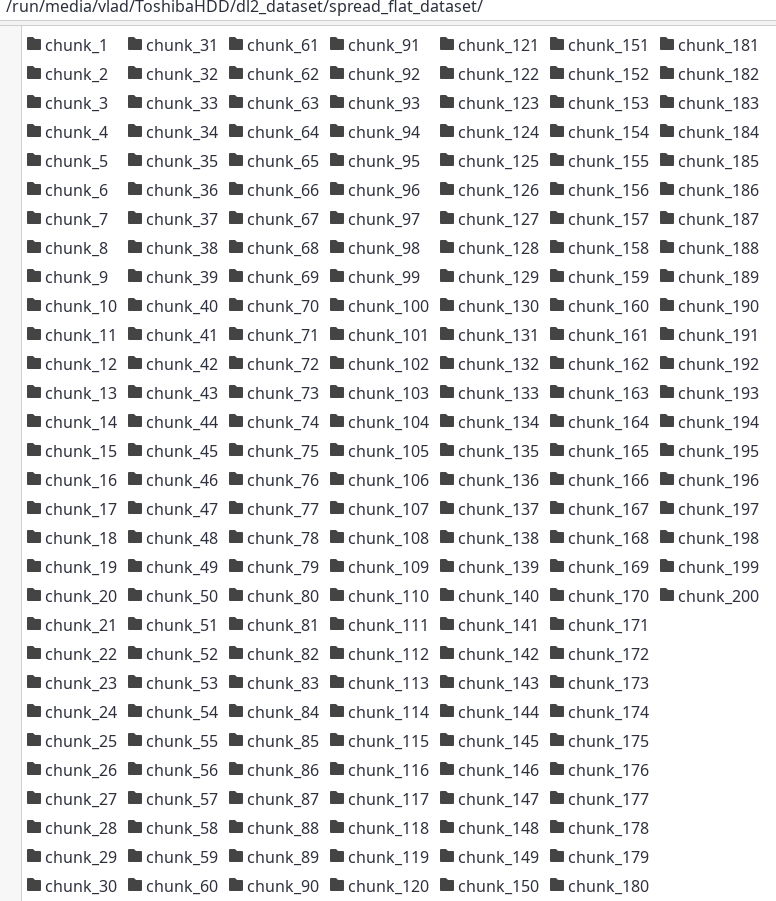
for ((i = 100; i < 150; i++)); do для копії spread\_out\_flat\_dataset\_3.sh

for ((i = 150; i < 200; i++)); do для копії spread\_out\_flat\_dataset\_4.sh

Тепер можна запускати усі ці скрипти одночасно, вони не перешкоджатимуть один одному, адже працюватимуть на своїх виділених частинах набору даних. Можна було обійтись і без копій, помістивши усю логіку в одному файлі, але так як цей скрипт використовавтиметься лише один раз у всьому процесі тренування, складність, що дадасть принцип DRY, не варта того.

Під час перенесення файлів у шматки, було помічено що у скрипта іноді виникають складнощі з файлами, що містять квадратні скобки, наприклад:

AUDIOBOOKS\_Василь\_Шкляр\_-\_Ключ\_[2007]\_wav\_spk\_id\_1816715\_16142579962243.wav

Для таких файлів скрипт не може додати відповідний запис у файлі транскрипцій. Для того, щоб вирішити цю проблему, зберемо усі файли з помилок, що були надруковані у консоль та вставимо їх у скрипт під назвою fix\_brackets\_issue\_at\_spreading.sh. Скрипт полягатиме у пошуку проходженні по усім проблемним файлам, правильній обробці квадратних скобок, знаходженні запису в файлі транскрипцій повного набору даних, та копіюванні його у файл транскрипції шматка. Після закінчення трансформації набору даних, spread\_flat\_dataset матиме наступний вигляд:

Далі, щоб завантажувати дані зі шматків для тренування в fine\_tune.py потрібно оновити скрипт, що відповідає за завантаження даних – flat\_dataset.py. Створимо новий скрипт spread\_flat\_dataset.py на його основі.

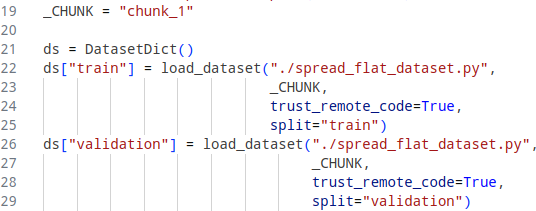
Головною відмінністю spread\_flat\_dataset.py від flat\_dataset.py буде те, що він матиме додатковий об’єкт конфігурації, шо буде вказувати яку саме підпапку-шматок ми намагаємося завантажити.

Тобто, усього ми матимемо 200 об’єктів конфігурацій – по одному для кожного шматка. Головні методи \_info, \_split\_generator та \_generate\_examples залишаться майже без змін: нам потрібно лише вказати назву підпапки-шматка у шляхах до даних використовуючи об’єкт конфігурацій:



У fine\_tune.py змінимо назву файла завантажувача даних з flat\_dataset.py на spread\_flat\_dataset.py. Також варто помітити, що під час тренування ми не використовуюємо піднабір даних для тестування test/test.tsv, тому завантажувати його в fine\_tune.py не має сенсу. Він буде потрібний в окремому скрипті для тестування точності моделі (eval\_on\_test.py), який ми створимо згодом.

Зміни в fine\_tune.py стосовно завантаження даних:



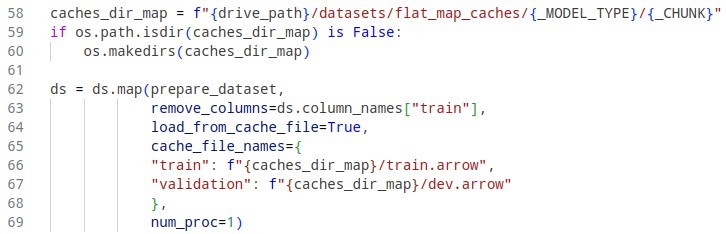
Тренувальні гіперпараметри також зазнають деяких змін. По-перше кількість кроків тренування зменшиться до приблизно 480 циклів (партій). Тому потрібно зменшити кількість циклів для розминки (warmup\_steps). По-друге, зручніше використовувати стратегію тренування epoch, замість steps. Epoch завершується коли всі дані тренувальної частини було пройдено один раз. Так як даних в нас дуже багато (відносно спроможностей), то кількість epoch можна зробити 1 (один), тобто тренування шматка завершується коли усі тренувальні дані пройшли через модель один раз. Після цього ми зберігаємо результат тренування параметрів моделі та рухаємося на наступний шматок. В нашому випадку краще протренувати додатковий шмат ніж протренувати якийсь шмат двічі. Нам важливіше аби модель побачила якомого більше екземплярів з різних джерел, ніж щоб модель сфокусувалася на якомусь одному джерелі.

План тренування такий: спочатку протренувати шматки 1-200 на epoch=1, потім, за потреби, повторити проходження ще декілька epoch.

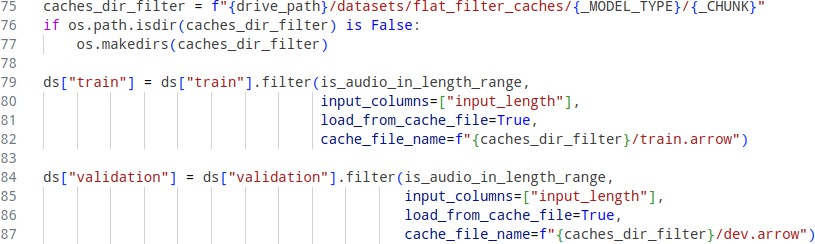
Повний список знін гіперпараметрів:

1. warmup\_steps=500 → 50 max\_steps=500 → num\_train\_epochs=1 ( 5 → 4 → 1)
2. evaluation\_strategy=“steps” → “epoch”
3. save\_steps=100, eval\_steps=100 → save\_strategy=“epoch”
4. load\_best\_model\_at\_end=True → False

Для того, щоб мати більше контролю за кешуванням prepare\_dataset (.map), додамо свої шляхи для кешу:



Зробимо теж саме для filter, хоча кешування цієї операції менш важливе:



Це всі зміни до fine\_tune.py файлу. Тепер ми готові порівняти швидкість тренування у локальному середовищі та Google Colab. Для одного шматка порівняння виглядає так:

|  |  |  |
| --- | --- | --- |
| Середовище | Завантаження даних | Тренування даних |
| Локальне | 1 хвилина | 5 годин 30 хвилин |
| Google Colab | 46 хвилин | 25 хвилин |

Бачимо, що завантаження даних, контрінтуїтивно, набагато швидше виконується у локальному середовищі. По-перше ця операція виконується на процесорі, а не на відеокарті. Також було помічено, що не зважаючи на num\_proc=1 параметр, локальний комп’ютер використовує паралелізацію на шести ядрах процесора. Ще можливо, що писання даних у файли кешу набагато швидше на локальній ssd, ніж на сервіс Google Drive.

З самим тренування не виникає сюрпризів – середовище Google Colab з відеокартою V100 тренує приблизно у 13 разів швидше ніж локальна відеокарта GTX 1660 Ti.

Виходячи з цих спостережень, тренування у хмарному середовищі є більше оптимальним варіантом. Також має сенс підготувати кеш для prepare\_dataset заздалегідь, тут є два варіанти:

1. створити кеш локально (у папці datasets/flat\_map\_caches) і потім завантажити вміст цієї папки до Google Drive
2. Створити кеш в окремому Colab Notebook процесі, що не буде використовувати ресурс відеокарти.

Для обох варіантів можна використати скрипт fine\_tune.py, що потрібно буде перервати після map(prepare\_dataset) операції.

Після створення кешу процес тренування полягає в наступному:

* + протренувати шмат N на epoch=1,
  + зберегти результат тренування на шматку N в папці models,
  + перейти до наступного шматка N+1
  + завантажити останню збережену версію моделі (з параметрами тренування N)
  + повторити

Нам вдалося протренувати модель на 100 шматках з 200. В таблиці нижче наведений результати тренування:

|  |  |  |
| --- | --- | --- |
| Шмат | Функця втрат | WER |
| 1 | 0.262 | 24.374 |
| 2 | 0.265 | 23.840 |
| 3 | 0.289 | 22.708 |
| 4 | 0.181 | 17.227 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Шмат | Функця втрат | WER | | |
| 5 | 0.315 | 24.669 | | |
| 6 | 0.283 | 25.932 | | |
| 7 | 1.887 | 37.786 | | |
| 8 | 2.505 | 38.129 | | |
| 9 | 1.492 | 31.764 | | |
| 10 | 0.216 | 24.825 | | |
| 11 | 0.216 | 25.938 | | |
| 12 | 0.191 | 24.678 | | |
| 13 | 0.451 | 24.642 | | |
| 14 | 0.266 | 18.046 | | |
| 15 | 0.183 | 22.039 | | |
| 16 | 0.181 | 22.080 | | |
| 17 | 0.151 | 21.221 | | |
| 18 | 0.263 | 26.232 | | |
| 19 | 0.214 | 18.917 | | |
| 20 | 0.166 | 18.949 | | |
| 21 | 0.171 | 19.327 | | |
| 22 | 0.182 | 22.046 | | |
| 23 | 0.558 | 37.291 | | |
| 24 | 0.486 | 21.254 | | |
| 25 | 0.190 | 21.193 | | |
| 26 | 0.176 | 21.679 | | |
| 27 | 0.192 | 21.717 | | |
| 28 | 0.221 | 23.967 | | |
| 29 | 0.398 | 21.790 | | |
| 30 | 0.342 | 21.228 | | |
| 31 | 0.252 | 20.916 | | |
| Шмат | Функця втрат | | | WER |
| 32 | 0.136 | | | 13.256 |
| 33 | 0.154 | | | 17.352 |
| 34 | 0.136 | | | 17.212 |
| 35 | 0.214 | | | 20.153 |
| 36 | 0.152 | | | 18.926 |
| 37 | 0.131 | | | 18.591 |
| 38 | 0.134 | | | 17.815 |
| 39 | 0.187 | | | 18.857 |
| 40 | 0.137 | | | 17.169 |
| 41 | 0.146 | | | 15.354 |
| 42 | 0.138 | | | 16.778 |
| 43 | 0.114 | | | 15.989 |
| 44 | 0.124 | | | 14.035 |
| 45 | 0.128 | | | 16.138 |
| 46 | 0.129 | | | 14.436 |
| 47 | 0.141 | | | 17.302 |
| 48 | 0.139 | | | 17.599 |
| 49 | 0.154 | | | 20.146 |
| 50 | 0.155 | | | 18.632 |
| 51 | 0.122 | | | 15.396 |
| 52 | 0.113 | | | 15.985 |
| 53 | 0.152 | | | 17.262 |
| 54 | 0.140 | | | 15.031 |
| 55 | 0.140 | | | 17.939 |
| 56 | 0.149 | | | 18.302 |
| 57 | 0.141 | | | 17.540 |
| 58 | 0.135 | | | 16.957 |
| Шмат | Функця втрат | | | WER |
| 59 | 0.163 | | | 17.523 |
| 60 | 0.384 | | | 25.403 |
| 61 | 0.481 | | | 33.905 |
| 62 | 0.204 | | | 22.219 |
| 63 | 0.148 | | | 15.419 |
| 64 | 0.164 | | | 18.832 |
| 65 | 0.264 | | | 20.159 |
| 66 | 0.263 | | | 28.571 |
| 67 | 0.244 | | | 27.782 |
| 68 | 0.169 | | | 21.795 |
| 69 | 0.170 | | | 18.199 |
| 70 | 0.185 | | | 22.429 |
| 71 | 0.156 | | | 17.833 |
| 72 | 0.154 | | | 18.622 |
| 73 | 0.182 | | | 19.179 |
| 74 | 0.259 | | | 24.731 |
| 75 | 0.441 | | | 38.869 |
| 76 | 1.168 | | | 60.637 |
| 77 | 1.009 | | | 59.993 |
| 78 | 0.191 | | | 21.131 |
| 79 | 0.158 | | | 18.872 |
| 80 | 0.148 | | | 16.362 |
| 81 | 0.223 | | | 29.820 |
| 82 | 0.185 | | | 21.838 |
| 83 | 0.167 | | | 17.741 |
| 84 | 0.152 | | | 18.654 |
| 85 | 0.152 | | | 17.093 |
| Шмат | Функця втрат | | WER | |
| 86 | 0.204 | | 24.867 | |
| 87 | 0.192 | | 21.195 | |
| 88 | 0.132 | | 18.003 | |
| 89 | 0.154 | | 18.964 | |
| 90 | 0.075 | | 8.416 | |
| 91 | 0.079 | | 8.289 | |
| 92 | 0.133 | | 15.259 | |
| 93 | 0.145 | | 17.686 | |
| 94 | 0.207 | | 24.688 | |
| 95 | 0.199 | | 23.939 | |
| 96 | 0.229 | | 26.199 | |
| 97 | 0.247 | | 30.492 | |
| 98 | 0.218 | | 24.669 | |
| 99 | 0.216 | | 27.541 | |
| 100 | 0.178 | | 22.715 | |

ВИСНОВКИ ДО РОЗДІЛУ 2

В цьому розділі був детально описаний процес створення набору даних для задачі автоматичного розпізнавання усної української мови. Були описані проблеми, що ми зустріли під час цього процесу, та їх вирішення.

Було описано такі поняття як попередньо навчена модель, техніка переносу навчання, ресурсність мови.

Була детально описана архітектура моделі Whisper, її переваги та недостатки, та чому саме ця модель була вибрана за базову.

Був повністю описаний процес створення тренувального середовища для моделі розпізнавання мовлення.

Однією з найбільших проблем під час цього процесу було керування величезним обсягом даних. Хоча наявність великого набору даних є перевагою у контексті узагальнення та стійкості моделі, це також створило значне обмеження на обчислювальні ресурси, та змусило шукати альтернативи локальним обчисленням і загалом оптимізовувати процеси тренування.

Було вирішено, що модель повинна побачити якомога більше даних, замість максимізації точності за допомогою великої кількості епох (epoch). Логіка цього рішення полягала в тому, що збільшення різноманітності даних, які бачить модель, ймовірно, більш сприятиме її навчанню, ніж перенавчення на меншій підмножині даних. Хоча менша кількість епох може обмежити глибину налаштування, ми прагнули, щоб модель зіткнулася з широким спектром варіацій української мови, акцентів та аудіоумов.

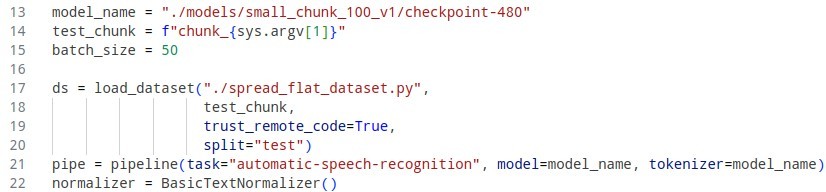
У підсумку модель було протреновано на 100 малих піднаборах даних (7,7 тисяч тренувальних екземплярів в піднаборі) по одному епоху на кожний піднабір. Результати тренування кожного епоху було занесено в таблицю для аналізу.

РОЗДІЛ 3. ПІСЛЯ ТРЕНУВАННЯ.

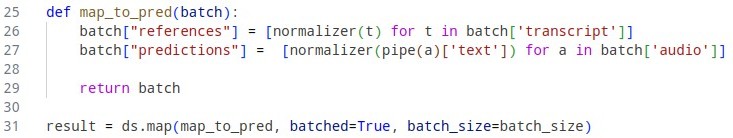
Аналіз результатів тренування моделі.

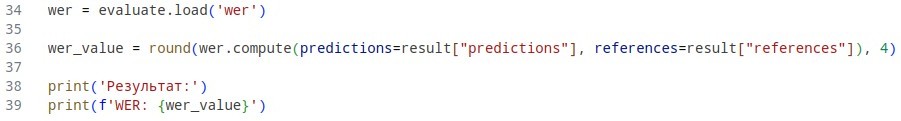
Модель може звикнути до тренувального та валідаційного піднаборів даних, тому для перевірки реальної точності моделі слід використати тестовий набір даних, що ніяким чином не був задіяний під час тренування.

Нагадаємо, що наш набір даних був занадто великим для обробки за раз, тому ми вирішили розбити його на піднабори – шматки. Ця техніка називається сегментацією. На виході ми отримали 200 малих наборів даних, які мають таку саму структуру що і початковий набір даних. У кожному з цих піднаборів є невеликий розподіл відведений для тестування. На них ми і перевіримо точність моделі за допомогою WER метрики. Для цього створимо скрипт під назвою eval\_on\_test.py.

У цьому скрипті нам потрібно вказати шлях до моделі, що ми будемо перевіряти. Ми перервали тренування на 100-ому шматку, тому ми вважатимемо модель збережену після цього кроку остаточною. Номер шматка для перевірки будемо приймати як перший аргумент до скрипта. Також нам потрібно завантажувати не всі дані, а тільки test розподіл. Для інстанціації моделі використаємо метод pipeline бібліотеки transformers платформи Hugging face. Код:

Далі підготуємо дані для оцінки:



Працюватимемо з партіями для швидшої оцінки даних. Для того, щоб підготувати партію екземплярів до оцінки WER потрібно вирахувати два значення: перше –reference – правильна відповідь, друге – prediction – результат з моделі. Останній крок – розрахування WER за допомогою бібліотеки evaluate:

Випадково виберемо декілька шматків для перевірки тестового розподілу і наведемо результати тренування у таблиці нижче:

|  |  |  |
| --- | --- | --- |
| Шмат | WER | Точність, % |
| 36 | 0.168 | 83.2 |
| 113 | 0.277 | 72.3 |
| 139 | 0.153 | 84.7 |
| 152 | 0.136 | 86.4 |
| 193 | 0.259 | 74.1 |
| 110 | 0.133 | 86.7 |
| 115 | 0.176 | 82.4 |
| 125 | 0.183 | 81.7 |
| 190 | 0.095 | 90.5 |
| 195 | 0.276 | 72.4 |

Отримаємо результати роботи базової моделі на тих самих тестових даних. Для цього скопіюємо контент скрипту eval\_on\_test.py в новий файл під назвою eval\_on\_test\_base\_m.py і змінемо модель з нашої натренованої на базову:

-model\_name = "./models/small\_chunk\_100\_v1/checkpoint-480"

+model\_name = "openai/whisper-small"

Результати базової моделі на тих самих тестових розподілах:

|  |  |  |
| --- | --- | --- |
| Шмат | WER | Точність, % |
| 36 | 1.037 | 0.0 |
| 113 | 0.570 | 43.0 |
| 139 | 0.968 | 3.2 |
| 152 | 1.228 | 0.0 |
| 193 | 1.729 | 0.0 |
| 110 | 0.927 | 7.3 |
| 115 | 1.361 | 0.0 |
| 125 | 1.816 | 0.0 |
| 190 | 0.888 | 11.2 |
| 195 | 3.156 | 0.0 |

З цих даних можна отримати наступні висновки.

1. Загальне покращення продуктивності

Наша модель суттєво перевершує базову модель Whisper на кожному шматку, демонструючи значно нижчий рівень WER.

В деяких випадках покращення дуже значне (наприклад, на 195-му шматку WER нашої моделі становить 0.276 порівняно з 3.156 у базової моделі Whisper). Це свідчить про те, що наша модель добре вивчила специфічні патерни цільової мови або підлаштувалася під дані.

1. Стабільність продуктивності

WER нашої моделі коливається від 0.095 до 0.277, медіаною близько 0.16 - 0.18, що свідчить про стабільний рівень продуктивності. Точність обернено-залежна від WER і коливається від 90.5% до 72.3%, медіаною близько 84 – 82%.

Водночас, у базової моделі Whisper спостерігається набагато більша варіативність результатів, де WER коливається від 0.57 до 3.16. Це вказує на те, що хоча Whisper є гарною моделлю загального призначення, вона не так добре справляється з нашим специфічним набором даних української мови. Аномальні результати базової моделі та варіативність нашої моделі

Існують набори даних, де базова модель показує значно гірші результати (наприклад, набори 125, 193, 195). Високі показники WER у цих випадках свідчать про складність окремих частин нашого набору даних. Вони можуть включати:

* + Акценти або діалекти, які Whisper не опрацювала під час навчання.
  + Шумні дані або незвичні лінгвістичні патерни
  + Рідкісна лексика або термінологія певної галузі.

Хоча WER нашої моделі загалом стабільний, на наборах 113, 193, 195 спостерігається певне збільшення помилок (0.277, 0.259, 0.276 відповідно). Це вказує на те, що модель також стикається з труднощами на цих ділянках, хоча ці помилки значно менші порівняно з базовою моделлю. Пізніше ми розглянемо вихідні послідовності нашої моделі та порівняємо їх з правильними відповідями вручну на прикладі 193-го шматка.

1. Покращення на складних даних

Там де базова модель модель зазнає найбільше труднощів (125, 193, 195), наша модель все одно підтримує відносно низкі показники WER. Це свідчить про те, що донавчання зробило нашу модель більш стійкою до складних випадків, де базова модель Whisper дає збій.

1. Співвідношення зниження WER

У більшості випадків наша модель досягає 70-90% зниження WER порівняно з базовою моделлю, що є значним покращенням. Наприклад:

* + Шматок 195: зниження на 91.25% (з 3.156 до 0.276)
  + Шматок 125: зниження на 89.92% (з 1.816 до 0.183)

Навіть у найгіршому випадку (шматок 113) спостерігається суттєве покращення: зниження на 51.4% (0.570 до 0.277).

Підсумок.

Наша модель показує високу продуктивність на різних тестових наборах, стабільно перевершуючи базову модель Whisper. Покращення найбільш помітне у складних випадках (набори 125, 193 і 195), що свідчить про те, що донавчання дозволило нашій моделі краще справлятися з мовними особливостями нашого датасету.

Незначна варіативність WER між наборами свідчить про те, що є ще деякі можливості для подальшого вдосконалення, особливо на наборах 113, 193 і 195.

На час оцінки тестового набору розмір партії дорівнює 50 екземплярів. Розглянемо вихідні послідовності нашої моделі (прогноз) та порівняємо їх з правильними відповідями (еталон) на одній із партій шматка 193. Наведемо значення у порівняльній таблиці:

|  |  |  |
| --- | --- | --- |
| № | Тип | Значення |
| 1 | Еталон | ринки особистому |
| 1 | Прогноз | ринки особистому |
| 2 | Еталон | з тридцять метрової висоти ейфелевої вежі ми побачили париж у рожевому серпанку з якого вимальовувалися собор паризької богоматері |
| 2 | Прогноз | з ми матері |
| 3 | Еталон | посвідчення |
| 3 | Прогноз | посвідчення |
| 4 | Еталон | про те що |
| 4 | Прогноз | про те що |
| 5 | Еталон | закінчилася |
| 5 | Прогноз | закінчилася |
| 6 | Еталон | продукт добору селекція |
| 6 | Прогноз | продукт добору селекція |
| 7 | Еталон | якщо морози |
| 7 | Прогноз | якщо морози |

|  |  |  |
| --- | --- | --- |
| № | Тип | Значення |
| 8 | Еталон | та морози якщо багнюка та |
| 8 | Прогноз | та морози якщо багинюка то |
| 9 | Еталон | життєрадісний |
| 9 | Прогноз | житті радісний |
| 10 | Еталон | це по науковому простіше кажучи |
| 10 | Прогноз | це по кажучи |
| 11 | Еталон | році купив вам |
| 11 | Прогноз | ж оця купив вам |
| 12 | Еталон | проблема буде |
| 12 | Прогноз | проблема буде |
| 13 | Еталон | ніякої америки вона |
| 13 | Прогноз | якої вона |
| 14 | Еталон | що гроші платять за |
| 14 | Прогноз | що гроші за |
| 15 | Еталон | цієї виставки |
| 15 | Прогноз | цієї виставки |
| 16 | Еталон | міститься |
| 16 | Прогноз | міститься |
| 17 | Еталон | штабу сумнозвісної організації нато |
| 17 | Прогноз | штаб нато |
| 18 | Еталон | що агресивне |
| 18 | Прогноз | що агресивне |
| 19 | Еталон | ось повністю |
| 19 | Прогноз | ось повністю |
| 20 | Еталон | зрозуміти |
| 20 | Прогноз | розуміти |
| 21 | Еталон | за допомогою трудової дисципліни |
| 21 | Прогноз | за допомогою трудової дисципліни |

|  |  |  |
| --- | --- | --- |
| № | Тип | Значення |
| 22 | Еталон | звання майстра золоті руки ні прогресивки ні |
| 22 | Прогноз | звання ні |
| 23 | Еталон | стала світовим лідером з виробництва відходів |
| 23 | Прогноз | стала світовим лідером з виробництва відходів |
| 24 | Еталон | літературний |
| 24 | Прогноз | літературний |
| 25 | Еталон | редактор режисер |
| 25 | Прогноз | редактор робанна мильниченку звуку режисер сергій вачі режисер |
| 26 | Еталон | володимир коваленко |
| 26 | Прогноз | володимир ковиденко |
| 27 | Еталон | частіше ставали предметом гордості місцевого масштабу яких завжди приємно побачити на екрані і все таки чутки про незвичайні рекорди записані в |
| 27 | Прогноз | щастіше ставили предметами гордості місцевого масштабу яких завжди приємно побачити на екрані і все таки чутки про незвучайні рекорди записані в |
| 28 | Еталон | книзі просочувалися |
| 28 | Прогноз | книзі просочувалися |
| 29 | Еталон | архиповича я вважаю що це чудово |
| 29 | Прогноз | архи повича я вважаю що це чудово |
| 30 | Еталон | знімали монтували цей матеріал упродовж |
| 30 | Прогноз | знімали імунтували цей матеріал упродовж |
| 31 | Еталон | в якій зовсім |
| 31 | Прогноз | в якийсь тільки зовсім |
| 32 | Еталон | комунізму |
| 32 | Прогноз | комунізму |
| 33 | Еталон | тисяча дев ятсот |
| 33 | Прогноз | тисяча дев ятсот |
| 34 | Еталон | дідусь ленін сказав що |
| 34 | Прогноз | дідус ленін сказав що |

|  |  |  |
| --- | --- | --- |
| № | Тип | Значення |
| 35 | Еталон | ще старанніше цю будівлю зводитимуть наші діти |
| 35 | Прогноз | ще стараніше цю будівлю зводитимуть наші діти |
| 36 | Еталон | кашпіровського вже близько |
| 36 | Прогноз | кашпіровського вже близько |
| 37 | Еталон | церковних книг |
| 37 | Прогноз | церковних книг |
| 38 | Еталон | товариша сталіна |
| 38 | Прогноз | товариша сталіна |
| 39 | Еталон | люди лише |
| 39 | Прогноз | люди лише |
| 40 | Еталон | і не подумайте |
| 40 | Прогноз | і не подумайте |
| 41 | Еталон | нашої якими |
| 41 | Прогноз | нашої країни перед якими |
| 42 | Еталон | своєю вірою анатолій анатолій михайлович кашпіровський любить повторювати слова поета |
| 42 | Прогноз | своєю вірою анатолій михайлович кашпировський любить повторювати слова поета |
| 43 | Еталон | тому їй приснився |
| 43 | Прогноз | тому пристився |
| 44 | Еталон | схожістю незмінним |
| 44 | Прогноз | схожестю незмінним |
| 45 | Еталон | варто порушувати |
| 45 | Прогноз | варто порушувати |
| 46 | Еталон | взагалі тому що розділити |
| 46 | Прогноз | взагалі тому що розділити |
| 47 | Еталон | до і допитливі |
| 47 | Прогноз | до і допитливі |

|  |  |  |
| --- | --- | --- |
| № | Тип | Значення |
| 48 | Еталон | дослідники дійшли висновку що організм людини неймовірно складний механізм з величезним потенціалом і можливості його |
| 48 | Прогноз | дослідники дійшли висновку що організм людини неймовірно складний механізм з величезним потенціалом і можливістя його |
| 49 | Еталон | сновидінь |
| 49 | Прогноз | сновидінь |
| 50 | Еталон | голос висоцького |
| 50 | Прогноз | голос висоцького |

Із цих результатів можна отримати наступні висновки.

1. Висока точність у простих, коротких фразах.

Коротші та простіші фрази, такі як "ринки особистому", "посвідчення", "про те що", "закінчилася", транскрибуються точно. Це свідчить про те, що модель добре працює з ізольованими та нескладними сегментами.

1. Проблеми з комплексними або довгими реченнями:

У довших реченнях модель має труднощі зі збереженням змісту та точності. Наприклад:

Еталон: "з тридцять метрової висоти ейфелевої вежі ми побачили париж у рожевому серпанку з якого вимальовувалися собор паризької богоматері" → Прогноз: "з ми матері".

Транскрипція неповна, втрачаються важливі частини речення, що призводить до значної втрати інформації.

1. Помилки на рівні морфем:

Деякі помилки пов’язані зі зміною морфем або незначними змінами структури слова, наприклад:

Еталон: "життєрадісний" → Прогноз: "житті радісний"

Оригінальне слово "життєрадісний" поділене на два слова, що змінює його значення.

1. Фонетична схожість, але семантичні помилки:

Деякі передбачення фонетично схожі на оригінал, але містять помилки, які змінюють зміст:

Еталон: "та морози якщо багнюка та" → Прогноз: "та морози якщо багинюка то". Слово "багнюка" (грязь) стає "багинюка", такого слова не існує.

1. Помилки з власними назвами та іменами:

Модель допускає помилки при роботі з власними назвами та іменами: Еталон: "володимир коваленко" → Прогноз: "володимир ковиденко".

У прізвищі "коваленко" модель допускає помилку, змінюючи його на "ковиденко".

1. Структурні пропуски:

Модель іноді пропускає слова або фрази, що призводить до неповних транскрипцій: Еталон: "штабу сумнозвісної організації нато" → Прогноз: "штаб нато".

Передбачення опускає "сумнозвісної організації", що змінює тон і зміст фрази.

1. Помилки з функціональними словами:

модель допускає помилки у функціональних або сполучних словах, які важливі для структури речення:

Еталон: "що гроші платять за" → Прогноз: "що гроші за".

Пропуск слова "платять" змінює зміст і робить речення менш зв'язним.

1. Збереження контексту в довгих реченнях:

У довших реченнях модель має труднощі зі збереженням повного контексту та логічної послідовності:

Еталон: "щастіше ставали предметом гордості місцевого масштабу" → Прогноз: "щастіше ставили предметами гордості місцевого масштабу".

Хоча кількість слів збережено, відбувається зміна дієслова та однина змінюється на множину, що впливає на загальний зміст.

Висновок:

Покращення можуть бути спрямовані на підвищення здатності моделі обробляти довші речення, точно розпізнавати власні назви та зберігати повну структуру речення, особливо у складній та офіційній мові.

Бенчмаркінг.

Є спеціальні набіри даних, що використовуються для бенчмаркінгу. Зазвичай вони відрізняються від простих наборів даних, тим що вони більш ретельно підготовані. Кожний екземпляр був перевірений компетентною людиною вручну, тобто відсутні автоматичні транскрипції (наприклад, якась інша модель ОПМ могла створити певну кількіть транскрипцій замість людини). Бенчмарковий набір даних дозволяє дослідникам і розробникам оцінювати точність і ефективність своїх моделей.

Він надає контрольоване середовище, де різні моделі можуть тестуватися на одному і тому ж наборі даних, що дозволяє робити послідовні порівняння між ними. Одним із таких наборів даних є mozilla common voice dataset. Для української мови він є чи не єдиним набором даних для бенчмаркінгу. На жаль, ми не можемо використати його для бенчмаркінгу. Все через те, що ми частково включили його до набору даних для тренування та валідації. Бенчмаркінг моделі на наборі даних, який частково використовувався для навчання, є проблематичним через ризик витоку даних, що може штучно підвищити показники продуктивності моделі. Навіть якщо окремі навчальні записи видалити, модель може навчитися розпізнавати шаблони або отримати упередження з подібних або пов’язаних даних під час навчання, що призведе до нереалістичної оцінки. Це ставить під сумнів достовірність результатів, оскільки модель може добре працювати на знайомих даних, але показати слабкі результати на справді нових прикладах. Для справедливої та неупередженої оцінки важливо використовувати повністю окремий набір даних.

Порівняння з іншими моделями, які використовують альтернативну архітектуру і добре себе зарекомендували. Однією з таких моделей є VOSK [59].

VOSK — це відкритий інструментарій для автоматичного розпізнавання мовлення. Він базується на традиційних методах, які часто поєднують акустичні моделі (наприклад, моделі суміші Гауса) з моделями прихованих марковських процесів.

Моделі VOSK відомі своєю легкістю, високою ефективністю та здатністю працювати на різних пристроях, включаючи вбудовані системи з обмеженими обчислювальними ресурсами.

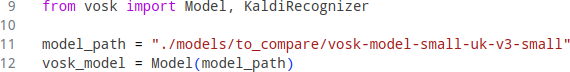
Хоча VOSK базується на традиційних методах, він використовує глибокі нейронні мережі для акустичного моделювання. Однак ці моделі не використовують архітектуру трансформер, яка часто зустрічається в сучасних системах розпізнавання мовлення, таких як Whisper.

Для української мови вже існує модель з архітектурою VOSK, використаємо її для порівняння з нашою моделлю. Проведемо порівняння на тих же тестових розподілах.

Створимо новий скрипт eval\_on\_test\_vosk.py. Щоб забезпечити однорідне середовище перевірки двох моделей зробимо цей скрипт максимально схожим на eval\_on\_test.py, з декількома необхідними змінами. По-перше, потрібно замінити нашу модель на VOSK, для завантаження якої нам знадобиться python пакет vosk.

По-друге, обробка даних на партії потребує змін: треба буде вручну зчитувати кожний аудіофайлів, перевіряти, чи відповідає він вимогам моделі (WAV, моно, PCM), і передавати аудіо дані частинами до розпізнавача (KaldiRecognizer) для транскрипції. Частина коду що відповідає за оцінку залишиться без змін.

Код завантаження моделі VOSK:



Код обробки даних на партії:



Результати роботи моделі VOSK наведено у таблиці нижче:

|  |  |  |
| --- | --- | --- |
| Шмат | WER | Точність, % |
| 36 | 0.277 | 72.3 |
| 113 | 0.267 | 73.3 |
| 139 | 0.294 | 70.6 |
| 152 | 0.309 | 69.1 |
| 193 | 0.328 | 67.2 |
| 110 | 0.171 | 82.9 |
| 115 | 0.299 | 70.1 |
| 125 | 0.352 | 64.8 |
| 190 | 0.167 | 83.3 |
| 195 | 0.918 | 8.2 |

Порівняємо результати роботи VOSK з результатами роботи нашої моделі. Хоча VOSK модель показує доволі непогані результати майже на усіх шматках, окрім 195,вона все ж поступається у точності нашій моделі. Наша модель, що базується на архітектурі Whisper, має стабільно нижчий коефіцієнт помилок слів (WER) у більшості випадків:

* 36: Модель Whisper (0.168) перевершує VOSK (0.277).
* 113: Обидві моделі працюють майже однаково, Whisper (0.277) та VOSK (0.267).
* 139: Whisper (0.153) значно кращий за VOSK (0.294).
* 152: Whisper (0.136) значно перевершує VOSK (0.309).
* 193: Whisper (0.269) показує кращий результат, ніж VOSK (0.328).
* 110: Whisper (0.133) кращий за VOSK (0.171), хоча різниця менша.
* 115: Whisper (0.176) кращий за VOSK (0.299).
* 125: Whisper (0.183) значно кращий за VOSK (0.352).
* 190: Whisper (0.095) перевершує VOSK (0.167).
* 195: Whisper (0.276) значно перевершує VOSK (0.918), особливо враховуючи дуже високий WER у VOSK на цьому підрозділі.

Модель на основі Whisper показує більш стабільну продуктивність, із значеннями WER у діапазоні від 0.095 до 0.277. Натомість модель VOSK має більший розподіл результатів — від 0.167 до 0.918. Модель VOSK також має аномально низьку продуктивність на шматку 195 (0.918), що свідчить про труднощі в певних тестових випадках у порівнянні з моделлю Whisper. Whisper перевершує VOSK завдяки своїй вдосконаленій архітектурі Transformer, яка добре захоплює далекі залежності та контекст, що робить її більш точною для завдань розпізнавання мовлення. VOSK в свою чергу використовує більш традиційну архітектуру: приховані моделі Маркова і потребує менше ресурсів і пам’яті для роботи.

Варто відмітити, що наша модель була спеціально налаштована на схожих даних під час тренування (fine-tuning), а VOSK не була. Це також могло вплинути на результат.

Розробка додатку (веб-сайту) для демонстрації роботи моделі.

***Архітектура.***

Додаток складається з двох частин.

Перша частина – веб-сервер next.js, що відповідає за інтерфейс додатку.

Next.js — це популярний фреймворк для React.js – JavaScript-бібліотеки для створення користувацьких інтерфейсів. Next.js призначений для створення веб- застосунків з серверною (SSR, Server-side rendering) та статичною генерацією (картинки, стилі, html тощо). Він спрощує процес розробки, надаючи структурований спосіб керування маршрутизацією, рендерингом та отриманням даних. Ключові особливості включають автоматичну маршрутизацію сторінок, серверний рендеринг, статичну генерацію сайтів, API маршрути та маршрутизацію на основі файлової структури. Next.js це надбудова на Node.js – середовища виконання, яке дозволяє запускати JavaScript код на сервері. Інші важливі технології, що використовує next.js включають:

● Webpack: Бандлер, який використовується для компіляції JavaScript модулів та оптимізації ресурсів.

● Babel: Компілятор JavaScript, який перетворює сучасний JavaScript у сумісні версії для різних браузерів.

Друга частина – веб-сервер flask, що керує моделлю глибинного навчання та відповідає за підготовку аудіофайлів та генерацію транскрипції для них.

Flask — це популярний мікрофреймворк для Python, який використовується для створення веб-застосунків. Він легкий і мінімалістичний, надаючи лише основні функції, необхідні для веб-розробки, з можливістю додавання розширень для складніших завдань. Flask дозволяє легко створювати веб-застосунки завдяки простому інтерфейсу для маршрутизації, обробки запитів та відповіді.

Ми не можемо обійтися без python, адже мова є стандартом для машинного навчання.

Щоб запустити додаток локально потрібно мати node/npm та python3 на комп’ютері та виконати наступні інструкції:

1. npm install

Завантажує та встановлює залежності, зазначені у файлі package.json, у проєкт.

2. python -m venv .venv

Створює віртуальне середовище Python у папці .venv, ізольоване від системних пакетів. Метою ізоляції є уникнення конфліктів між пакетами та версіями залежностей у різних проєктах, забезпечуючи кожному проєкту власне середовище.

3. source .venv/bin/activate

Активує віртуальне середовище Python, перенаправляючи використання команд Python та pip на ізольовані версії всередині цього середовища.

4. pip install -r requirements.txt

Встановлює всі пакети та залежності, перелічені у файлі requirements.txt, у поточне середовище Python.

5. npm run dev

Запускає скрипт з назвою "dev", визначений у секції "scripts" файлу package.json, зазвичай для запуску локального сервера для розробки.

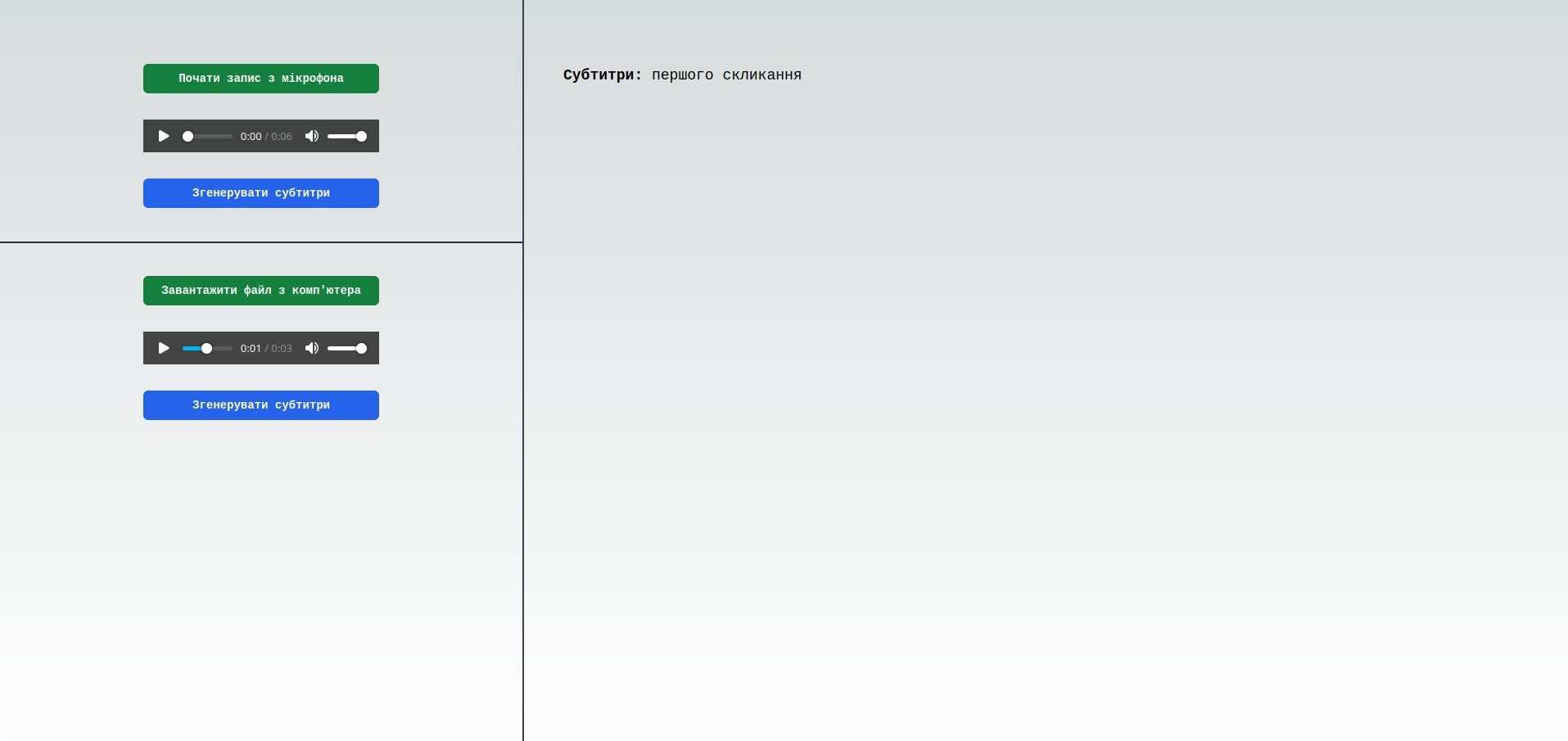
6. Відкрити http://localhost:3000 у браузері

Скрипт під назвою dev визначений у файлі package.json, нижче наведений код, що стосується цього скрипта (з секції scripts):



Cкрипт dev налаштований для одночасного запуску сервера розробки Next.js (порт 3000) і сервера розробки Flask (порт 5328) за допомогою пакету concurrently. Також скрипт автоматично переконується, що всі потрібні python пакети для роботи з моделлю встановлено.

Додаток складається з однієї веб-сторінки:



Сторінка містить секцію для введеня даних користувача зліва, та секцію для виведення результату генерації транскрипції моделлю справа.

Секція для введення даних містить дві підсекції:

1. для запису нового аудіофайлу за допомогою мікрофона
2. для завантаження існуючого аудіофайлу з будь-якої папки комп’ютера

Код сторінки:



LeftSide/RightSide компоненти розбивають сторінку на ліву та праву секції. AudioCaptureInputs компонент відповідає за запис нових аудіофайлів за допомогою мікрофона.

AudioUploadInputs компонент відповідає за завантаження існуючих аудіофайлів з комп’ютера.

useTranscribe – це хук, що відповідає за комунікацію з flask сервером, де розгорнута модель: передачу йому аудіофайлу в правильному форматі на ендпойнт /api/transcribe (POST) та отримання транскрипції для переданого аудіофайлу.

Розглянемо AudioCaptureInputs, AudioUploadInputs та useTranscribe більш детально. Обидва компоненти приймають метод transcribe, що визначений в useTranscribe і має наступну сигнатуру:

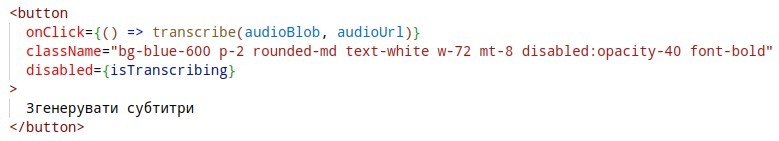


Тобто на вхід він приймає два параметри:

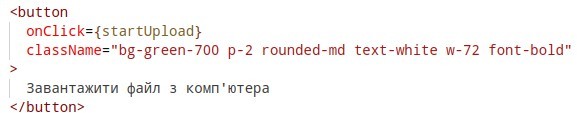
* перший у форматі Blob, що використовується для репрезентації файлів
* другий у форматі строки, тимчасовий шлях до файлу в браузері.

Обидва ці значення ми отримаємо під час завантаження/запису аудіофайлу

Після того як ми отримали ці значення, метод transcribe використовуватиметься однаковим чином що у компоненті запису нового аудіофайлу (AudioCaptureInputs), що у компоненті завантаження існуючого файлу (AudioUploadInputs):



Розглянемо як відбувається завантаження існуючого файлу (AudioUploadInputs). Процес завантаження починається з натискання кнопки:



Це призведе до виклику метода startUpload з хука useAudioUpload, що відповідає за обробку завантаженого файлу:



Можна бачити що тут використовується стандартний API браузера, а саме oninput аттрибут <input /> тега типу “file”. Обробник методу oninput буде викликаний коли користувач вибере файл для завантаження з комп’ютера

Завантажений файл матиме формат Blob.

Метод createObjectURL створює тимчасову URL-адресу для доступу до завантаженого аудіофайлу, що зберігається в пам'яті браузера.

Тепер розглянемо як відбувається запис нового аудіофайлу (AudioCaptureInputs). Процес запису починається з натискання кнопки:



Це призведе до виклику методу startRecording з хука useAudioCapture, що відповідає за запис файлу за допомогою мікрофона:



Суть метода полягає в отриманні дозволу від користувача на використання мікрофона за допомогою методу:

const stream = await navigator.mediaDevices.getUserMedia(...

Після отримання дозволу ми починаємо записувати аудіо за допомогою класу MediaRecorder:

const mediaRecorder = new MediaRecorder(stream, {…

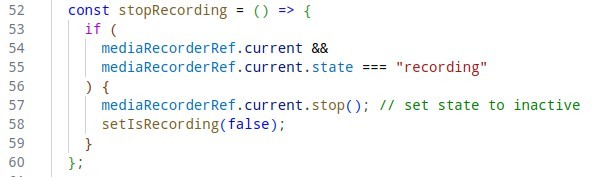
… mediaRecorder.start();

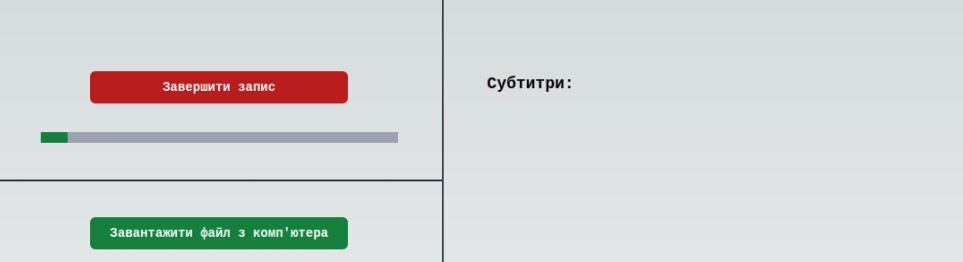
Дані запису будуть надходити до обробника dataavailable події об’єкту mediaRecorder шматками, де ми будемо їх записувати у спеціальних масив шматків, а при закінченні запису ми створимо з усіх отриманих шматків об’єкт типу Blob.

let blob = new Blob(chunksRef.current, { type: mimeType });

Увесь процес запису аудіофайлів покладається на стандартні API браузера, не потрібно жодних сторонніх бібліотек.

Для того, що зупинити запис потрібно викликати stop() метода об’єкту mediaRecorder. Для цього створено окремий метод stopRecording:



Цей метод буде викликатися при натисканні кнопки, що показуватиметься тільки під час запису аудіофайлу:

Розглянемо що саме робить метод transcribe:



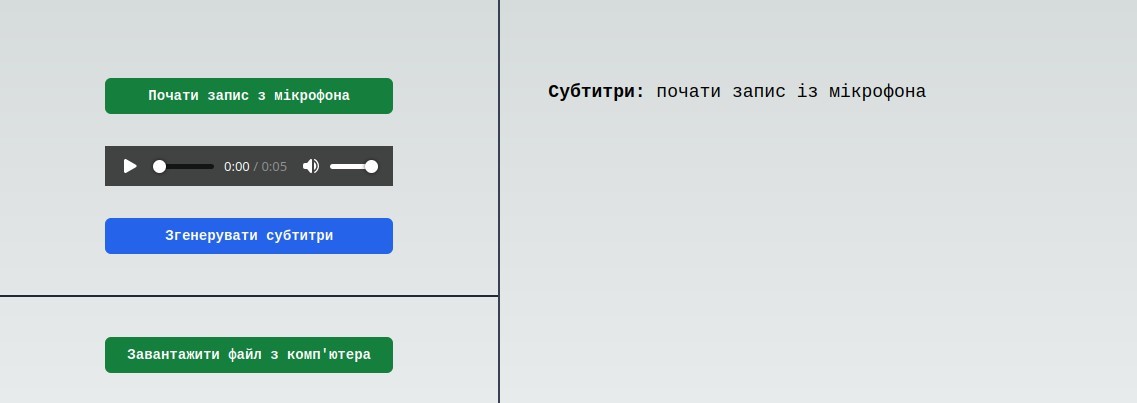
Метод transcribe поміщає Blob аудіофайлу в об’єкт FormData, що складається з пар ключів-значень, і відправляє його у запиті до /api/transcribe ендпойнту. Відповідь на запит міститиме інформацію про транскрипцію аудіофайлу в полі transcript. Метод fetch, що використовується створення запиту є стандартним API браузера.

Дуже важливу роль відіграє метод fixSampleRate, з імені стає зрозуміло, що метод виправляє семпл рейт (частоту дискретизації) аудіофайлу. Архітектура моделі Whiser, що використовується нами як базова модель, приймає тільки аудіофайли з частотою 16kHz. Під час тренування ми контролювали частоту дискретизації усіх екземплярів на кроці завантаження даних, але зараз ми приймаємо аудіофайли від користувача, і ці файли можуть мати будь-яку частоту, наприклад 8kHz, 44kHz, 48kHz. Важливо притримуватися однієї частоти щоб аудіофайли не звучали для моделі пришвидшено або уповільнено, бо це значно погіршить точність.

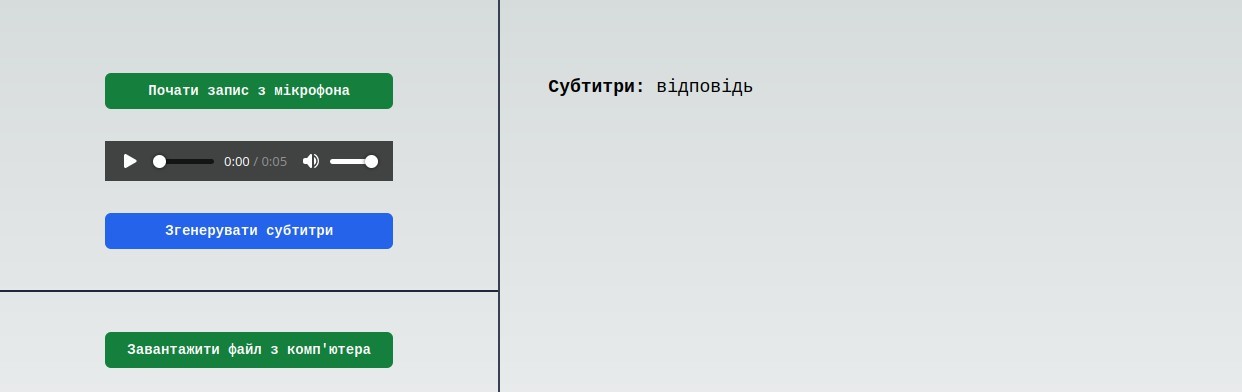
Продемонструємо цю проблему на прикладі.

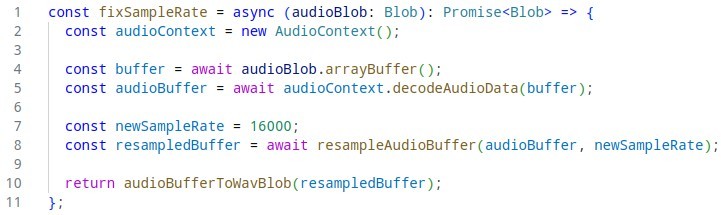
Запишемо аудіофайл з наступною фразою: “Почати запис із мікрофона”. З методом fixSampleRate частота дискретизації записаного файлу дорівнює 16kHz: [1] 2024-09-23 12:22:51: generate\_transcript sample\_rate: 16000

І модель видає коректну транскрипцію:



Якщо ж метод fixSampleRate тимчасово прибрати, то можна побачити, що насправді оригінальна частота дискретизації записаного аудіофайлу дорівнює 48kHz: [1] 2024-09-23 12:23:33: generate\_transcript sample\_rate: 48000

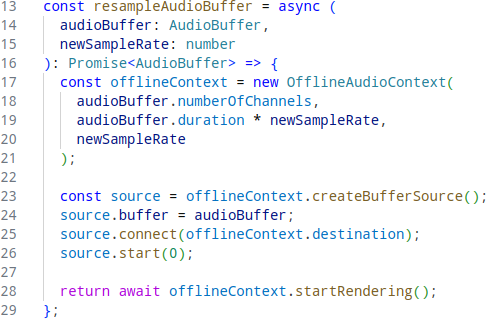
І якщо залишити цю частоту незмінною, то модель видасть невірну транскрипцію:

Розглянемо як fixSampleRate змінює частоту дискретизації з 48kHz та 16kHz. Основний код:

Спочатку метод трансформує дані з типу Blob до типу ArrayBuffer, що містить дані у бінарному форматі.

За допомогою decodeAudioData методу AudioContext бінарні дані з ArrayBuffer конвертуються у аудіоформат придатний для використання – AudioBuffer.

Далі ми змінюємо фактичну частоту дискретизації на бажану(16000) в функції resampleAudioBuffer. За цим переводимо дані з ArrayBuffer типу назад в Blob у audioBufferToWavBlob методі.

Найбільш важливою для нас є resampleAudioBuffer функція:

OfflineAudioContext — це спеціальний тип AudioContext, призначений для обробки аудіо офлайн, для обробки аудіо швидше за реальний час. Цей контекст ініціалізується трьома аргументами:

1. Кількість каналів (моно або стерео)
2. Загальна кількість фреймів, які будуть оброблятися. Розраховується як добуток тривалості та бажаної частоти дискретизації.
3. Бажана частота дискретизації

Це налаштовує середовище, яке може обробляти аудіо з новою частотою дискретизації без обмежень реального часу.

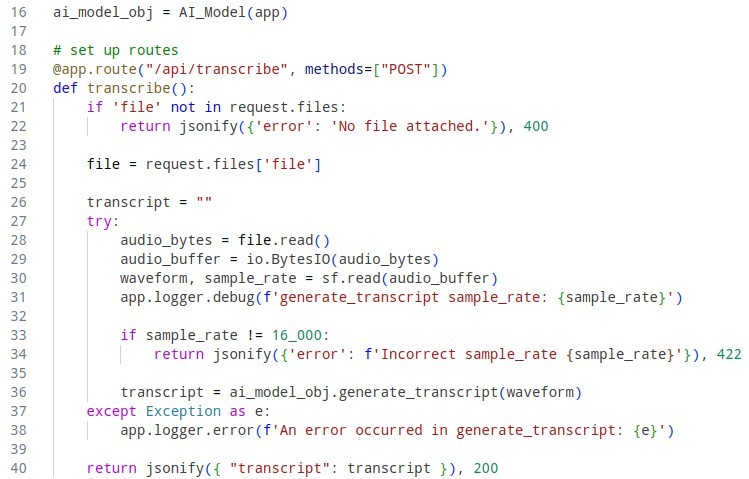
Далі відбувається створення та налаштування аудіо-джерела:

* + Створюється джерело буфера (createBufferSource), яке дозволяє відтворювати наданий audioBuffer.
  + source.buffer = audioBuffer: Оригінальний AudioBuffer призначається джерелу, щоб його можна було відтворювати.
  + source.connect(offlineContext.destination): Джерело підключається до пункту призначення offlineContext, який обробляє кінцевий аудіовихід.
  + source.start(0): Це запускає відтворення audioBuffer негайно, з моменту часу 0.

Фінальна операція функції - Рендеринг аудіо з новою частотою дискретизації.

offlineContext.startRendering() запускає процес офлайн рендерингу. Це перетворює аудіо та обробляє його відповідно до налаштувань offlineContext (тобто з новою частотою дискретизації). На вихід отримаємо новий AudioBuffer з бажаною частотою дискретизації.

Повернемося до методу transcribe і розглянемо обробник запиту на генерацію транскрипції, а саме /api/transcribe ендпойнт.



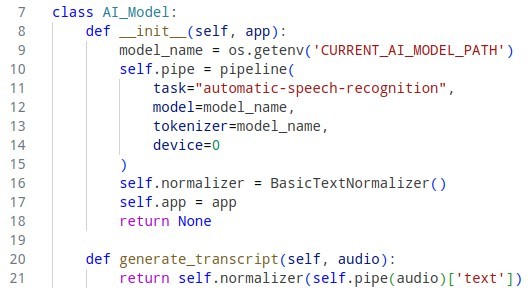
Під час запуску сервер завантажує модель для розпізнавання усного мовлення за допомогою класу AI\_Model.

Під час запиту обробник переконується, що file поле існує, і далі оброблює його:

* + file.read(): зчитує дані в бінарному форматі.
  + io.BytesIO(audio\_bytes): переводить бінарні дані у AudioBuffer формат.
  + sf.read(audio\_buffer): бібліотека soundfile декодує AudioBuffer і отримає форму хвилі та частоту дискретизації.
  + Обробник перевіряє чи відповідає частота дискретизації необхідній (16kHz).

На останок, обробник передає форму хвилі аудіо до generate\_transcript методу об’єкту моделі для отримання транскрипції та відправляє відповідь на запит

Клас AI\_Model, що представляє модель виглядає наступним чином:



Конструктор класу завантажує модель за допомогою pipeline методу з бібліотеки transformers платформи Hugging face. Для посилання на актуальну модель використовується змінна середовища (з файлу .env). Можна використати локальну модель вказавши шлях до неї (1) або вказати ключ моделі якщо вона завантажена на платформу Hugging face (2):

1. CURRENT\_AI\_MODEL\_PATH=ai\_models/small\_chunk\_100\_v1/checkpoint-480
2. CURRENT\_AI\_MODEL\_PATH=mXfdzNrud9fPMgJJsn/uni\_dl2\_asr

Конструктор також ініціалізує BasicTextNormalizer, щоб очищати транскрипцію після прогнозу, видаленням зайвих символів чи знаків.

generate\_transcript метод приймає форму хвилі аудіо і повертає очищену транскрипцію.

В цьому проекті ми успішно інтегрували систему автоматичного розпізнавання мовлення за допомогою платформи Hugging face та нашої натренованої моделі. Проект продемонстрував основи того, як інтегрувати модель у python проект. Також було показано як можна обробляти користувацькі аудіодані у формат зрозумілий для моделі. Цей проєкт може слугувати основою для більш просунутих застосунків автоматичного розпізнавання мовлення, таких як сервіси перетворення мовлення в текст, програми з голосовим керуванням або аналіз аудіо-даних. Подальші покращення можуть включати інтеграцію додаткових мов, вдосконалення відновлення після помилок та оптимізацію для обробки більших наборів даних.

Рекомендації щодо продовження тренування моделі.

Тренувати модель лише на 1 епох недостатньо. Для покращення точності моделі потрібно спробувати пройти ті самі дані ще декілька разів. Не обов’язково використовувати усі шматки з 1 по 100. Варто передивитися результати тренування (точність на валідаційному наборі) та зосередитися на місцях де результат був найгіршим. Протренувати такі набори додатково на 1-5 епох. Перевірити чи покращився результат. Кожний шматок має невеликий тестовий розпопіл, варто час від часу робити перевірки і на ньому.

Також слід задіяти усі дані. Наразі нам вдалося протренувати лише половину від повного набору даних: 100 з 200 шматків. Потрібно розширити тренування моделі на шматки 101-200. Спочатку тренуватимемо на 1 епох, і якщо на якомусь із шматків модель видаватиме незадовільний результат, то слід для цього набору збільшити кількість епох.

Також можна спробувати змінити деякі гіперпараметри, та подивитися як це вплине на результат тренування. Наприклад, взяти нашу нинішню модель, вибрати для неї якийсь важкий шматок, і протренувати його на кілька епохів (1-3), зберегти нову модель. Потім, змінити певний гіперпараметр, наприклад learning\_rate (швидкість навчання) і повторити той самий процес. В результаті ми отримаємо декілька паралельних моделей і зможемо оцінити, котра з них досягла кращого результату під час короткого тренування. З цієї оцінки слідуватиме яке значення гіперпараметру буде використане для подальшого тренування. Для процесу оцінки варто буде також використати тестовий розподіл, для більшої впевненості.

Так як наша модель має демо сторінку, слід використати це для онлайн-навчання. Онлайн-навчання означає процес, де модель оновлюється поступово, коли стають доступними нові дані, замість того, щоб тренуватися в одному великому блоці на статичному наборі даних. Модель навчається на одному або кількох прикладах одночасно і постійно оновлюється, коли надходять нові дані. Онлайн-навчання дозволяє моделям адаптуватися до змін у розподілі даних або навколишньому середовищі, що робить його підходящим для динамічних, реальних застосувань. Онлайн-навчання має низку переваг порівняно з традиційним (офлайновим) процесом навчання:

● Персоналізація: в системах розпізнавання мовлення онлайн-навчання можна використовувати для адаптації моделі до голосу та акценту конкретного користувача з часом. Коли система отримує більше мовленнєвих даних від користувача, вона може поступово краще розпізнавати мовні патерни цього користувача.

● Адаптація до нової лексики в реальному часі: Онлайн-навчання може допомогти моделям ASR динамічно адаптуватися до нових слів, сленгу або технічних термінів, які не були присутні у тренувальних даних. Наприклад, віртуальний асистент може навчитися розпізнавати нові імена або технічні терміни під час використання.

● Адаптація до середовища: Моделі ASR можуть адаптуватися до різного фону шуму або акустичних умов (наприклад, різних кімнат, зовнішнього середовища), навчаючись на нових аудіоданих у реальному часі.

● Низькоресурсне та безперервне навчання: У контексті мов з обмеженими ресурсами або недопредставлених діалектів онлайн-навчання може бути корисним. Модель може почати з невеликого набору даних і продовжувати навчатися у міру появи нових даних, уникаючи необхідності великих початкових наборів даних.

Онлайн навчання може бути реалізоване за допомогою різних підходів, одним із найбільш популярних і простих є стохастичний градієнтний спуск. Перевага цього методу для онлайн-навчання полягає в тому, що він дозволяє працювати з індивідуальними екземплярами і не потребує формування партій. Під час онлайн- навчання модель постійно отримує нові дані, і їй потрібно оновлюватися, не чекаючи повного набору даних.

Онлайн-навчання є корисною технікою, проте слід пам’тати і про недоліки:

● Катастрофічне забування: Якщо модель постійно оновлюється без збереження знань про минулі дані, вона може "забути" раніше вивчену інформацію. Це є серйозною проблемою, особливо для нейронних мереж.

● Стабільність моделі: Оскільки модель оновлюється на основі окремих або невеликих партій даних, існує ризик нестабільності, якщо дані містять шум або упередження..

Якщо онлайн-навчання негативно впливає на точність моделі через недоліки, що були наведені вище, можна спробувати альтернативний підхід, що є більш безпечним – поступове (інкрементальне) навчання. Він полягає у зберіганні екземплярів, що модель зустріла онлайн, і формуванні з них нових міні-наборів даних. Модель почне тренування на нових даних лише тоді, коли міні-набор матиме достатній розмір і буде перевірений людиною (бажано). Тобто тренування моделі відбуватиметься не відразу, як прибув нових екземпляр, а з регулярним інтервалом на більшій кількості екземплярів за раз. Це має покращити стабільність моделі, та зробити її більш стійкою до забування старих даних.

Коли модель протренується достатньо разів на наявних даних, можна буде розглянути техніки нарощення даних (data augmentation). Нарощення даних передбачає штучне розширення навчального набору даних шляхом створення варіацій існуючих даних. Ці методи допомагають покращити стійкість моделі, підвищуючи її здатність адаптуватися до різних умов, акцентів і шумів, що робить модель більш пристосованою до реальних сценаріїв. Найбільш популярні техніки нарощення даних для автоматичного розпізнавання мовлення включають:

● Зміна швидкості аудіо зі збереженням висоти тону. Наприклад, аудіо може бути уповільнене або прискорене на невеликий відсоток (наприклад, 90%, 100%, 110%). Це допомагає моделі узагальнювати дані для різних темпів мовлення. Є найбільш популярною технікою через свою простоту та ефективність.

● Зсув висоти тону зі збереженням швидкості аудіо. Інверс попередньої техніки, допомагає моделі краще справлятися з варіаціями у висоті голосу, що корисно для роботи з відмінностями у голосах чоловіків і жінок або змінами голосу. Менш поширена.

● Додавання фонового шуму до аудіо, наприклад звуків вулиці, білого шуму або гаміру натовпу. Робить модель більш стійкою до шуму навколишнього середовища, підвищуючи її продуктивність в таких умовах. Використовується часто.

● Імітація акустики приміщення. Полягає у імітації акустичних ефектів різних середовищ (наприклад, великого залу або маленької кімнати) шляхом застосування фільтрів до аудіо. Допомагає моделі справлятися з різними ефектами реверберації, спричиненими різними приміщеннями. Часто використовується для моделей, що працюють у різних середовищах.

Зміна довжини голосового тракту. Полягає у імітації різної довжини голосового тракту шляхом деформації осі частоти аудіо. Допомагає моделі краще узагальнювати дані для мовців з різною довжиною голосового тракту (наприклад, дітей та дорослих). Менш поширена

Як саме впроваджувати нарощування даних? Для цього можна використати сторонні бібліотеки мови python, наприклад, librosa. Наведемо декілька прикладів для технік нарощування, що були згадані вище:

Зміна швидкості аудіо зі збереженням висоти тону:

Зсув висоти тону зі збереженням швидкості аудіо:



Додавання фонового шуму до аудіо:

Дотримуючись цих рекомендацій ми зможемо значно покращити точність нашої моделі та зробити її більш стійкою до нових, раніше не бачених даних та середовищ

ВИСНОВКИ ДО РОЗДІЛУ 3

В цьому розділі був детально описаний процес аналізу результатів моделі на тестових даних. Для цього була вирахована точність моделі на тестових розподілах 10 шматків (~10 тисяч екземплярів загалом).

Ми також оцінили результати базової моделі та порівняли наскільки тренування на нашому наборі даних допомогло покращити показники точності. Показник WER був знижений на 70-90% після тренування.

Під час оцінки кожного шматка ми розбивали дані на партії по 50 екземплярів для того, щоб скористатися паралелізмом. На одній з цих партій були продемонстровані реальні вихідні значення моделі в порівнянні з еталонними значеннями. Також проведений аналіз недоліків результатів моделі.

Потім ми порівняли точність нашої моделі на тестових даних з результатами існуючої моделі VOSK, представника класичної архітектури – прихованих моделях Маркова. Наша модель перевершила модель VOSK на майже усіх ділянках. Середній WER нашої моделі на 15% нижчий за VOSK.

Значна частина розділу була присвячена створенню веб-додатку для демонстрації можливостей моделі. Ми помістили модель на окремий flask сервер та робили запити на нього для транскрибування завантажених аудіофайлів. Користувач мав змогу завантажити наявний файл з комп’ютера або записати новий за допомогою мікрофона через API браузера. Особливу увагу було приділено важливості відповідності між частотою дискретизації аудіофайла та очікуванням моделі.

Останній підрозділ був присвячений стратегіям тренування для подальшого покращення точності моделі. Найбільш дієвими рекомендаціями є розширення тренування на усі доступні дані, тренування складних для моделі ділянок на додаткову кількість епох, налаштування таких гіперпараметрів, як швидкість навчання (learning\_rate). Також була представлена стратегія для тренування моделі на нових даних під назвою онлайн-навчання, її переваги та недоліки, а також альтернатива до неї – інкрементальне навчання, якщо недоліки занадто негативно впливають на модель.

В кінці була запропонована техніка на майбутнє – нарощення даних, особливо корисна коли даних для подальшого тренування стане недостатньо. Також були наведені практичні рекомендації (код) щодо реалізації цієї техніки

ВИСНОВКИ

Обробка природної мови полягає в пошуку зв’язків між складовими частинами мови. Важливі етапи: попередня обробка даних, вилучення ознак, моделювання.

Попередня обробка ознак – переведення сирих даних у формат, що модель може зрозуміти. Моделі зазвичай з певним фіксованим значенням частоти дискретизації.

Вилучення ознак – перетворення аудіосигналів у набір значеннєвих представлень (ознак), які модель використовуватиме для розпізнавання та інтерпретації мовлення. Допомагає позбавитися некорисних даних, зменшити вимірність даних. До популярних технік належать: мел-частотні кепстральні коефіцієнти та перцептивне лінійне передбачення. Обидва методи ґрунтуються на тому, як люди сприймають звук, і роблять аудіосигнал більш схожим на те, як його сприймає людський слух, зменшуючи вплив частот, менш важливих для слуху, і підсилюючи значущі для слуху частоти.

Вилучені ознаки використовуються як вхідні дані. Вилучені ознаки часто нормалізуються перед використанням для покращення навчання моделі. Популярними методами нормалізації для моделей автоматичного розпізнавання мовлення є нормалізація z-показника та нормалізація довжини голосового тракту. Нормалізація за z-показником перетворює значення в наборі даних так, щоб вони мали середнє 0 і стандартне відхилення 1. Це дозволяє порівнювати значення на різних шкалах, роблячи їх співмірними. Нормалізація довжини голосового тракту – це метод, який допомагає зробити мовлення різних людей більш схожим, навіть якщо їхні голосові тракти різної довжини. Він змінює частотні характеристики мовлення, щоб компенсувати відмінності між голосами.

Нині для завдання автоматичного розпізнання усної мови є актуальними дві категорії моделей: традиційні методи машинного навчання та методи глибинного навчання. До популярних традиційних методів можна віднести приховані моделі Маркова та моделі суміші Гауса. ПМП моделюють послідовність звуків мови (фонем) як серію станів, де кожен стан представляє сегмент мовного сигналу.

Модель суміші Гауса є імовірнісною моделлю для представлення нормально розподілених субпопуляцій у загальній сукупності. Приклад: МСГ може описати запис мовлення як поєднання двох Гаусових розподілів, де один представляє тиху фонему з низькою інтенсивністю звуку, а інший — голосну фонему з вищою інтенсивністю, дозволяючи оцінювати, до якої фонеми належить певний звуковий фрагмент. Часто МСГ використовується разом з ПМП: Кожний стан у ПМП представлений за допомогою моделі суміші Гауса, яка моделює розподіл ймовірностей.

Рекурентні нейронні мережі — це клас штучних нейронних мереж, у якому з'єднання між вузлами утворюють граф орієнтований у часі. Це створює внутрішній стан мережі, що дозволяє їй проявляти динамічну поведінку в часі. Переваги: можливість обробки вхідних даних будь-якої довжини, розмір моделі не збільшується разом з розміром вхідних даних, розрахунок враховує історичну інформацію, ваги повторно використовуються на кожному кроці часу t. Недоліки: повільне обчислення, труднощі з доступом до давньої інформації, явище зникнення/вибуху градієнта, неможливо розглядати майбутній вхід (не проблема для двонаправлених РНН).

Довга короткочасна пам'ять (ДКЧП, long short-term memory, LSTM) — це тип рекурентної нейронної мережі (РНМ), спрямованої на вирішення проблеми зникнення/вибуху градієнта, присутньої в традиційних РНМ. Блок ДКЧП складається з клітини (запам’ятовує значення протягом довільних проміжків часу), вхідного вентиля (вирішує, які частини нової інформації зберігати в поточному стані), вихідного вентиля (контролює, які фрагменти інформації в поточному стані виводити) та забувального вентиля (вирішує, яку інформацію відкинути з попереднього стану призначаючи значення від 0 до 1). Вибіркове виведення відповідної інформації дозволяє ДКЧП підтримувати корисні довгострокові залежності для прогнозування.

Seq2Seq моделі використовують ДКЧП і мають змогу зіставляти вхідні послідовності однієї довжини з вихідними послідовностями іншої довжини. Модель seq2seq складається з кодера, вектора кодера (state) та декодера. Кодер фіксує контекст вхідної послідовності та надсилає його в декодер, який потім створює остаточну вихідну послідовність

Механізм уваги дозволяє декодеру «переглядати» різні частини вхідної послідовності на кожному кроці генерації вихідного стану. Важливо те, що модель може дізнатися, на що звернути увагу, на основі вхідної послідовності та того, що вона сгенерувала до цього часу.

Трансформер уникає рекурентності і натомість повністю покладається на механізм самоуваги. Цей механізм дозволяє розглядати усі слова вхідної послідовності одночасно, що пришвидшує навчання. Самоувага наповнює модель контекстуальною обізнаністю, що, в свою чергу, дозволить моделі розрізняти значення окремих елементів у послідовності та динамічно регулювати їхній вплив на кінцевий результат. Механізм самоуваги часто розширюється до механізму багатоголової уваги, що використовує кілька вивчених наборів трансформацій, або «голов».

Самоувага не фіксує порядок елементів у вхідній послідовності, щоб усунути це обмеження використовується позиційні кодування.

Архітектура трансформера складається з кодера та декодера. Завдання кодера полягає в тому, щоб зіставити вхідну послідовність з послідовністю неперервних представлень, які потім подаються в декодер. Декодер отримує вихідні дані кодера разом із вихідними даними свого попередньго блоку та генерує вихідну послідовність.

Перший крок створення моделі – збір набору даних для тренування моделі. Поєднання декількох різних наборів даних допоможе моделі уникнути перенавчення (overfitting), адже дані належать різним джерелам і мовникам. Для полегшення тренування структуру комбінованого набору даних потрібно зробити більш плоскою.

За основу для нашої моделі ми візьмемо попередньо навчену модель та скористаємося технікою перенесення навчання. Мотивацію перенесення навчання можна знайти в ідеї «Вчимося вчитися» (NIPS 95), що стверджує що навчання tabula rasa часто обмежене. Природні мови для задачі автоматичного розпізнавання мовлення розрізняють на дві категорії: мови з високим ресурсом та мови з низьким ресурсом (за кількістю даних для тренування в свободному доступі). Англійська є прикладом мови з високим ресурсом, українська – з низьким. В нашому випадку перенесення навчання буде відбуватися з англійської мови на українську.

Архітектура трансформер була представлена в 2017 році, в науковій публікації Attention Is All You Need (Ashish Vaswani et al., “Увага це все що вам потрібно”). Трансформери витіснили багато попередніх підходів до багатьох завдань машинного навчання та стали основною нейронною архітектурою для задачі автоматичного розпізнавання мовлення.

Одним з найкращих представників архітектури трансформер у галузі автоматичного розпізнавання мовлення є модель компанії OpenAI під назвою Whisper (WSPSR – Web-scale Supervised Pre-training for Speech Recognition). Ця модель є сильним кандидатом для роботи з мовами з низьким ресурсом, як-от українська, з низки переваг наведених нижче:

* Whisper навчається на великому різноманітному наборі даних (680,000 годин), що включає багато мов. Це дозволяє моделі досить добре працювати навіть на мовах з малою кількістю даних.
* Використовується метод слабкого керування.
* Whisper може виконувати розпізнавання навіть на нових для нього мовах (zero- shot learning). Це дозволяє моделі краще узагальнювати мови, використовуючи знання зі споріднених мов.
* Whisper розроблений бути стійким проти діалектів та акцентів.

Архітектура Whisper реалізована за допомогою трансформера, що складається з кодера та декодера. Модель підтримує частоту дискретизації аудіодоріжки 16kHz, дуже важливо щоб вхідні дані дотримувалися цієї вимоги. Whisper тренований на аудіофайлах довжиною до 30 секунд і не може приймати більш довгі дані. Під час тренування аудіодоріжки з довжиною, що перевищує цей ліміт, будуть відкидатися. Whisper має 5 конфігурацій (tiny, base, small, medium, large), що відрізняються за розміром: кількістю тренувальних параметрів, кількістю шарів, шириною шарів, кількістю голов механізму самоуваги. Для нашої моделі виберемо конфігурацію small, що вважається золотою серединою, адже вона має відносно малий розмір, але, назважаючи на це, справляється зі складними задачами, такими як, шум та акценти.

Мова програмування python – стандарт для будь-яких задач машинного навчання. Для полегшення процесу тренування моделі буде використана платформа Hugging face

Процес виконання завдання автоматичного розпізнавання мовлення можна розділити на 3 кроки:

1. Виявлення особливостей та при-обробка аудіо даних за допомогою екстрактора.
2. Відображення послідовності звуку в послідовність тексту (транскрипція почутого) – моделювання.
3. Відображення вихідної послідовності токенів у текстове представлення за допомогою токенізатора.

Для того, щоб почати тренувати модель у своєму скрипті потрібно реалізувати наступне:

1. Завантажити набір даних та провести попередню обробку.
2. Перетворити завантажені та попередньо оброблені дані в формат, що є зручним для тренування моделі - PyTorch тензори.
3. Визначити метрику оцінки: під час тренування використовуватиметься метрика під назвою частота помилки слова (Word rate error – WER). Чим нижчий показник WER тим вища точність моделі. WER також дозволяє проаналізувати, який саме тип помилки є слабкою стороною моделі: заміни, видалення чи вставки.
4. Завантажити попередньо навчену модель з певної контрольнох точки (checkpoint).
5. Визначити налаштування тренування за допомогою гіперпараметрів.
6. Виконати тренування: пропустити через модель набір даних певну кількість циклів, одну партію за раз, розраховуючи функцію втрат, та регулюючи параметри моделі після кожної партії екземплярів.
7. Перевірити результат тренування на тестовому розподілі набору даних – відкладеній частині тренувального набору даних, щоб дізнатися наскільки добре модель опановує нові дані.

Даний набір даних складається з приблизно півтора мільйонів екземплярів, але виявилося, що локально тренувати навіть невеликий піднабір цих даних не є посильною задачею через брак потужностей. Було вирішено перенести тренування у хмарне середовище, адже воно пропонує потужні відеокарти, що спеціалізуються на машинному навчанні і мають тензорні ядра

Також було вирішено розбити набір даних на піднабори – шматки. Їх вийшло рівно 200 (по ~7,7 тисяч тренувальних екземплярів, ~1 тисячі валідаційних та ~1 тисячі тестових екземплярів). Розбиття даних пришвидшить навчання та додасть контролю над ним. За час тренування ми встигли протренувати модель на 100 з 200 шматків (~850 тисяч екземплярів) на 1 епох.

Після тренування модель була перевірена на тестових розподілах 10ти шматків (~10 тисяч екземплярів) і порівняна з базовою моделлю Whisper. Наша модель суттєво перевершує базову модель Whisper на кожному шматку, демонструючи значно нижчий рівень WER. WER нашої моделі коливається від 0.095 до 0.277, медіаною близько 0.16 - 0.18, що свідчить про стабільний рівень продуктивності. Точність обернено-залежна від WER і коливається від 90.5% до 72.3%, медіаною близько 84 – 82%. Водночас, у базової моделі Whisper спостерігається набагато більша варіативність результатів, де WER коливається від 0.57 до 3.16. Наша модель краще пристосована до акцентів та специфічної лексики.

Була використана частина набору даних mozilla common voice під час тренування, тому він не може бути використаним для бенчмаркінгу.

Нашу модель було порівняно з уже існуючою моделлю VOSK для українського мовлення, що використовує більш традиційний архітектурний підхід – приховані моделі Маркова поєднані з моделями суміші Гауса.

VOSK модель показала доволі непогані результати майже на усіх шматках, але вона все ж поступається у точності нашій моделі, що має стабільно нижчий коефіцієнт помилок слів (WER) у більшості випадків і показує більш стабільну продуктивність, із значеннями WER у діапазоні від 0.095 до 0.277. Натомість модель VOSK має більший розподіл результатів — від 0.167 до 0.918. Варто відмітити, що наша модель була спеціально налаштована на схожих даних під час тренування (fine-tuning), а VOSK не була.

Для демонстрації роботи моделі був розроблений веб-застосунок. Він складається з двох частин: веб-серверу next.js, що відповідає за інтерфейс додатку та веб-серверу flask, що керує моделлю глибинного навчання для генерації транскрипції вхідного аудіофайлу

Веб-застосунок дозволяє завантажити існуючий аудіофайл з комп’ютера або записати новий у браузері за допомогою мікрофона та згенерувати транскрипцію почутого. Особлива увага була звернена на важливість дотримання значення частоти дискретизації, яке очікує модель (16kHz). Якщо частота дискретизації аудіофайлу буде відрізнятися від очікуваного, модель не зможе згенерувати коректну транскрипцію. У веб-застосунку було продемонстровано як можна інтегрувати модель машинного навчання у проект. Цей проєкт може слугувати основою для більш просунутих застосунків автоматичного розпізнавання мовлення.

На останок були запропоновані рекомендації з продовження тренування моделі.

1. Найбільш дієвим наступним кроком є задіяння усіх наявних даних для тренування (ще 100 шматків на 1 епох).
2. Тренувати на однин епох недостатньо в деяких випадках. Варто передивитися результати тренування (точність на валідаційному наборі) та зосередитися на місцях де результат був найгіршим. Протренувати такі набори додатково на 1-5 епох.
3. Рекомендується замінити деякі гіперпараметри, наприклад швидкість навчання, під час тренування одного шматку, щоб подивитися як це скажеться на результаті.
4. Так як наша модель має демо сторінку, слід використати це для онлайн-навчання, або його альтернативу – інкрементальне навчання.
5. Коли набір даних буде вичерпано, можна розглянути техніки нарощування даних, наприклад, зміну швидкості аудіо зі збереженням висоти тону, зсув висоти тону зі збереженням швидкості аудіо та додавання фонового шуму до аудіо.

Дотримуючись цих рекомендацій наша модель зможе покращити точність та стати більш стійкою до нових даних та середовищ

СПИСОК ДЖЕРЕЛ

1. Min Xu et al., HMM-Based Audio Keyword Generation, 2004.
2. [https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B](https://uk.wikipedia.org/wiki/Алгоритм_Баума_—_Велша) [8%D1%82%D0%BC\_%D0%91%D0%B0%D1%83%D0%BC%D0%B0\_%E2%80%94\_%D0%92%D0%B5%D0%BB%D1%88%D0%B0](https://uk.wikipedia.org/wiki/Алгоритм_Баума_—_Велша)
3. [https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B](https://uk.wikipedia.org/wiki/Алгоритм_Вітербі) [8%D1%82%D0%BC\_%D0%92%D1%96%D1%82%D0%B5%D1%80%D0%B1%D1%96](https://uk.wikipedia.org/wiki/Алгоритм_Вітербі)
4. Dong Wang et al., An Overview of End-to-End Automatic Speech Recognition, 2019.
5. Amir Hussein et al., Arabic Speech Recognition by End-to-End, Modular Systems and Human, 2021.
6. Mark Gales et al., The Application of Hidden Markov Models in Speech Recognition, 2008.
7. <https://en.wikipedia.org/wiki/Mixture_model>
8. <https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning>
9. <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>
10. [https://ocw.mit.edu/courses/18-409-algorithmic-aspects-of-machine-learning-spring-201](https://ocw.mit.edu/courses/18-409-algorithmic-aspects-of-machine-learning-spring-2015/e339520c4069ca5e785b29a3c604470e_MIT18_409S15_chapp6.pdf)

[5/e339520c4069ca5e785b29a3c604470e\_MIT18\_409S15\_chapp6.pdf](https://ocw.mit.edu/courses/18-409-algorithmic-aspects-of-machine-learning-spring-2015/e339520c4069ca5e785b29a3c604470e_MIT18_409S15_chapp6.pdf)

1. [https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D0%](https://uk.wikipedia.org/wiki/Рекурентна_нейронна_мережа) [B5%D0%BD%D1%82%D0%BD%D0%B0\_%D0%BD%D0%B5%D0%B9%D1%80%](https://uk.wikipedia.org/wiki/Рекурентна_нейронна_мережа) [D0%BE%D0%BD%D0%BD%D0%B0\_%D0%BC%D0%B5%D1%80%D0%B5%D0](https://uk.wikipedia.org/wiki/Рекурентна_нейронна_мережа)

[%B6%D0%B0](https://uk.wikipedia.org/wiki/Рекурентна_нейронна_мережа)

1. <https://www.deeplearning.ai/resources/natural-language-processing/> 13)[https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%BE%D0%B1%D0%](https://uk.wikipedia.org/wiki/Обробка_природної_мови)

[BA%D0%B0\_%D0%BF%D1%80%D0%B8%D1%80%D0%BE%D0%B4%D0%BD%](https://uk.wikipedia.org/wiki/Обробка_природної_мови)

[D0%BE%D1%97\_%D0%BC%D0%BE%D0%B2%D0%B8](https://uk.wikipedia.org/wiki/Обробка_природної_мови)

1. [https://uk.wikipedia.org/wiki/%D0%94%D0%BE%D0%B2%D0%B3%D0%B0\_%D0%](https://uk.wikipedia.org/wiki/Довга_короткочасна_пам%27ять) [BA%D0%BE%D1%80%D0%BE%D1%82%D0%BA%D0%BE%D1%87%D0%B0%D](https://uk.wikipedia.org/wiki/Довга_короткочасна_пам%27ять) [1%81%D0%BD%D0%B0\_%D0%BF%D0%B0%D0%BC'%D1%8F%D1%82%D1%8](https://uk.wikipedia.org/wiki/Довга_короткочасна_пам%27ять) [C](https://uk.wikipedia.org/wiki/Довга_короткочасна_пам%27ять)
2. Yoshua Bengio et al., Learning long-term dependencies with gradient descent is difficult, 1994.
3. Sepp Hochreiter and Jurgen Schmidhuber, Long Short-Term Memory, 1997.
4. Hassim Sak et al., Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling, 2014.
5. Felix A. Gers et al., Learning to Forget: Continual Prediction with LSTM, 2000.
6. Alex Graves et al., A Novel Connectionist System for Unconstrained Handwriting Recognition, 2009
7. Alex Graves et al., Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures, 2005.
8. Klaus Greff et al., LSTM: A Search Space Odyssey, 2015.
9. Felix A. Gers et al., Learning to Forget: Continual Prediction with LSTM, 2000.
10. Wojciech Zaremba Recurrent Neural Network Regularization, 2014.
11. Ilya Sutskever et al., Sequence to Sequence Learning with Neural Networks, 2014.
12. [https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-netw](https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/)

[orks-experts/](https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/)

1. Kyunghyun Cho et al., Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, 2014.
2. [https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-](https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346) [model-679e04af4346](https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346)
3. <https://dennybritz.com/posts/wildml/attention-and-memory-in-deep-learning-and-nlp/>
4. Dzmitry Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate, 2014.
5. Kelvin Xu et al., Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, 2015.
6. Misha Denil et al., Learning where to Attend with Deep Architectures for Image Tracking, 2011.
7. Karl Moritz Hermann, et al., Teaching Machines to Read and Comprehend, 2015.
8. Minh-Thang Luong et al., Effective Approaches to Attention-based Neural Machine Translation, 2015.
9. https://armanasq.github.io/nlp/self-attention/
10. https://machinelearningmastery.com/the-transformer-model/
11. https://d2l.ai/chapter\_attention-mechanisms-and-transformers/transformer.html
12. Rohit Prabhavalkar et al., End-to-End Speech Recognition: A Survey, 2023.
13. Tomas Mikolov et al., Efficient Estimation of Word Representations in Vector Space, 2013.
14. Awni Hannun et al., Deep Speech: Scaling up end-to-end speech recognition, 2014.
15. William Chan et al., Listen, Attend and Spell, 2015.
16. Tomas Mikolov et al., Distributed Representations of Words and Phrases and their Compositionality, 2013.
17. Ashish Vaswani et al., Attention Is All You Need, 2017.
18. Jacob Devlin et al., BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.
19. Dong Wang et al., Transfer Learning for Speech and Language Processing, 2015.
20. Alec Radford et al., Robust Speech Recognition via Large-Scale Weak Supervision (Whisper), 2022.
21. Bengio et al., A neural probabilistic language model, 2003.
22. [https://github.com/egorsmkv/speech-recognition-uk?tab=readme-ov-file#-datasets](https://github.com/egorsmkv/speech-recognition-uk?tab=readme-ov-file&-datasets)
23. <https://en.wikipedia.org/wiki/Whisper_(speech_recognition_system)>
24. <https://openai.com/index/whisper/>
25. <https://github.com/openai/whisper>
26. [https://uk.wikipedia.org/wiki/%D0%A1%D0%BB%D0%B0%D0%B1%D0%BA%D0%](https://uk.wikipedia.org/wiki/Слабке_керування)

[B5\_%D0%BA%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%](https://uk.wikipedia.org/wiki/Слабке_керування)

[D1%8F](https://uk.wikipedia.org/wiki/Слабке_керування)

1. Dan Hendrycks et al., A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks, 2016.
2. Rewon Child et al., Generating Long Sequences With Sparse Transformers, 2019.
3. Press & Wolf, Using the Output Embedding to Improve Language Models, 2017.
4. [https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BC%D0%B5%D0%](https://uk.wikipedia.org/wiki/Променевий_пошук) [BD%D0%B5%D0%B2%D0%B8%D0%B9\_%D0%BF%D0%BE%D1%88%D1%83%](https://uk.wikipedia.org/wiki/Променевий_пошук) [D0%BA](https://uk.wikipedia.org/wiki/Променевий_пошук)
5. <https://huggingface.co/blog/fine-tune-whisper>
6. <https://huggingface.co/learn/audio-course/chapter5/fine-tuning>
7. https://research.google/blog/specaugment-a-new-data-augmentation-method-for- automatic-speech-recognition/
8. <https://alphacephei.com/vosk/models>